PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION International Bureau



INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51)-International Patent Classification 5:

G06F 15/332

(11) International Publication Number: WO 94/23385

A2

(43) International Publication Date: 13 October 1994 (13.10.94)

(21) International Application Number:

PCT/GB94/00677

(22) International Filing Date:

30 March 1994 (30.03.94)

(30) Priority Data:

040,301 100,747 30 March 1993 (30.03.93)

30 July 1993 (30.07.93)

3) US US

(71)(72) Applicants and Inventors: LEWIS, Adrian, Stafford [GB/ES]; Federico Garcia Lorca 17-5-B, E-07014 Palma (ES). KNOWLES, Gregory, Percy [AU/ES]; Calle Menorca 18-2-B, E-07011 Palma (ES).

(74) Agent: JONES, Ian; W.P. Thompson & Co., Celcon House, 289-293 High Holborn, London WClV 7HU (GB).

(81) Designated States: AT, AU, BB, BG, BR, BY, CA, CH, CN, CZ, DE, DK, ES, FI, GB, HU, JP, KP, KR, KZ, LK, LU, LV, MG, MN, MW, NL, NO, NZ, PL, PT, RO, RU, SD, SE, SI, SK, TT, UA, UZ, VN, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

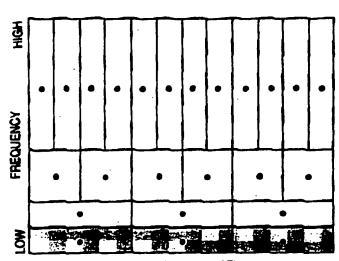
Published

Without international search report and to be republished upon receipt of that report.

(54) Title: DATA COMPRESSION AND DECOMPRESSION

(57) Abstract

A compression and decompression method uses a wavelet decompositin. frequency based tree encoding, tree based motion encoding, frequency weighted quantization, Huffman encoding, and/or tree based activity estimation for bit rate control. Forward and inverse quasi-perfect reconstruction transforms are used to generate the wavelet decomposition and to reconstruct data values close to the original data values. The forward and inverse quasi-perfect reconstruction transforms utilize special filters at the boundaries of the data being transformed and/or inverse transformed. Structures and methods are disclosed for traversing wavelet decompositions. Methods are disclosed for increasing software execution speed in the decompression of video. Fixed or variable length tokens are included in a compressed data stream to indicate changes in encoding methods used to generate the compressed data stream.



TIME (OR SPACE)

LOCALITY 3

LOW PASS
COMPONENT

HIGH PASS
COMPONENT

DATA VALUE

This Page Blank (uspto)

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

ΑT	Austria	GB	United Kingdom	MOR	Mauritania
ΑŪ	Australia	GE	Georgia	MEW	Malewi
88	Barbados	GN	Guines	NB	Niger
BE	Belgium	GR	Greece	NL	Netherlands
BF	Burkina Paso	HU	Hungary	NO	Norway
BG	Bulgaria	IE	Ireland	NZ	New Zealand
BJ	Benin	π	(taly	PL	Poland
BR	Brazil	JP	Japan	PT	Portugal
BY	Belarus	KE	Kenya	RO	Romania
CA	Canada	KG	Kyrgystan	RU	Russian Federation
CF	Central African Republic	KP	Democratic People's Republic	SD	States
CG	Congo		of Korea	SE	Sweden
CH	Switzerland	KR	Republic of Korea	SI	Slovenia
a	Côte d'Ivoire	KZ	Kazakh-stan	SK	Slovakta
CM	Салогоо	ഥ	Liechtenstein	SN	Sonegal
CN	China	LK	Sri Lanka	TD	Chad
C2	Czechoslovakia	LU	Luxembourg	TG	Togo
CZ	Czech Republic	LV	Latvia	· TJ	Tajikistan
DE	Germany	MC	Monaco	TT	Trinidad and Tobago
DE	Denmark	MD	Republic of Moldova	UA	Ukraine
ES	Spain	MG	Madagascar	US	United States of America
FT	Finland	ML	Mali	UZ	Uzbekistan
FR	Prace	MIN	Mongolia	٧N	Viet Nam
GA	Gehon				

DATA COMPRESSION AND DECOMPRESSION

CROSS REFERENCE TO APPENDICES

Appendix A, which is a part of the present disclosure, is a listing of a software implementation written in the programming language C.

Appendices B-1 and B-2, which are part of the present disclosure, together are a description of a hardware 10 implementation in the commonly used hardware description language ELLA.

Appendix C, which is part of the present disclosure is a listing of a software implementation written in the programming language C and assembly code.

A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document, but otherwise reserves all copyright rights whatsoever.

20 FIELD OF THE INVENTION

This invention relates to a method of and apparatus for data compression and decompression. In particular, this invention relates the compression, decompression, transmission and storage of audio, still-image and video data in digital form.

BACKGROUND INFORMATION

An image such as an image displayed on a computer monitor may be represented as a two-dimensional matrix of digital data values. A single frame on a VGA computer 30 monitor may, for example, be represented as three matrixes of pixel values. Each of the three matrixes has a data value which corresponds to a pixel on the monitor.

The images on the monitor can be represented by a 640 by 480 matrix of data values representing the luminance

(brightness) values Y of the pixels of the screen and two other 640 by 480 matrixes of data values representing the chrominance (color) values U and V of the pixels on the screen. Although the luminance and chrominance values are analog values, the one luminance value and the two chrominance values for a pixel may be digitized from analog form into discrete digital values. Each luminance and chrominance digital value may be represented by an 8-bit number. One frame of a computer monitor therefore typically requires about 7 megabits of memory to store in an uncompressed form.

In view of the large amount of memory required to store or transmit a single image in uncompressed digital form, it would be desirable to compress the digital image data before storage or transmission in such a way that the compressed digital data could later be decompressed to recover the original image data for viewing. In this way, a smaller amount of compressed digital data could be stored or transmitted. Accordingly, numerous digital image compression and decompression methods have been developed.

According to one method, each individual digital value is converted into a corresponding digital code. Some of the codes have a small number of bits whereas 25 others of the codes have a larger number of bits. In order to take advantage of the fact that some of the codes are short whereas others of the codes are longer, the original digital data values of the original image are filtered using digital filters into a high frequency component and 30 a low frequency component. The high frequency component represents ambiguities in the image and is therefore observed to have a comparatively large number of identical data values for real-world images. By encoding the commonly occurring digital data values in the high 35 frequency component with the short digital codes, the total number of bits required to store the image data can be reduced from the number of bits that would otherwise be

required if 8-bits were used to represent all of the data values. Because the total number of bits in the resulting encoded data is less than the total number of bits in the original sequence of data values, the original image is said to have been compressed.

To decompress the compressed encoded data to recover the original image data, the compressed encoded data is decoded using the same digital code. The resulting high and low frequency components are then recombined to form the two-dimensional matrix of original image data values.

Where the data being compressed is two-dimensional data such as image data, separation of the original data into high and low frequency components by the digital filters may be accomplished by filtering in two dimensions such as the horizontal dimension of the image and the vertical dimension of the image. Similarly, decoded high and low frequency components can be recombined into the original image data values by recombining in two dimensions.

- To achieve even greater compression, the low frequency component may itself be filtered into its high and low frequency components before encoding. Similarly, the low frequency component of the low frequency component may also be refiltered. This process of recursive
- 25 filtering may be repeated a number of times. Whether or not recursive filtering is performed, the filtered image data is said to have been "transformed" into the high and low frequency components. This digital filtering is called a "transform". Similarly, the high and low pass
- 30 components are said to be "inverse transformed" back into the original data values. This process is known as the "inverse transform".

Figure 1 is a diagram of a digital gray-scale image of a solid black square 1 on a white background 2 35 represented by a 640 by 480 matrix of 8-bit data luminance values.

Figure 2 is a diagram illustrating a first

WO 94/23385 PCT/GB94/00677

- 4 -

intermediate step in the generation of the high and low frequency components of the original image. A high pass digital filter which outputs a single data value using multiple data values as inputs is first run across the 5 original image values from left to right, row by row, to generate G subblock 3. The number of digital values in G subblock 3 is half of the number of data values in the original image of Figure 1 because the digital filter is sequentially moved to the right by twos to process two 10 additional data values for each additional one data output generated for G subblock 3. Similarly, a low pass digital filter which outputs a single data value using multiple data values as inputs is first run across the original image values from left to right, row by row, to generate H 15 subblock 4. The number of digital values in H subblock 4 is half of the number of data values in the original image because the digital filter is moved to the right by twos to process two additional data values for each additional one data output generated for H subblock 4. Each of the 20 two vertical bars in high pass G subblock 3 appears where a change occurs spatially in the horizontal dimension in the original image of Figure 1. Where the G filter encounters a change from white data values to black data values when the filter G is run across the image of Figure 25 1 in a horizontal direction, the G digital filter outputs a corresponding black data value into subblock 3. Similarly, when the G digital filter encounters the next change, which is this time a change from black to white data values, the G digital filter again outputs a 30 corresponding black data value into G subblock 3.

Figure 3 is a diagram illustrating a second intermediate step in the generation of the high and low frequency components of the original image. The high pass digital filter is run down the various columns of the subblocks H and G of Figure 2 to form the HG subblock 5 and GG subblock 6 shown in Figure 3. Similarly, the low pass digital filter is run down the various columns of the

H and G subblocks 3 and 4 of Figure 2 to form HH and GH subblocks 7 and 8 shown in Figure 3. The result is the low pass component in subblock HH and the three high pass component subblocks GH, HG and GG. The total number of high and low pass component data values in Figure 3 is equal to the number of data values in the original image of Figure 1. The data values in the high pass component subblocks GH, HG and GG are referred to as the high frequency component data values of octave 0.

10 The low pass subblock HH is then filtered horizontally and vertically in the same way into its low and high frequency components. Figure 4 illustrates the The data values in HHHG subblock 9, resulting subblocks. HHGH subblock 10, and HHGG subblock 11 are referred to as 15 the high frequency component data vales of octave 1. Subblock HHHH is the low frequency component. Although not illustrated, the low frequency HHHH subblock 12 can be refiltered using the same method. As can be seen from Figure 4, the high frequency components of octaves 0 and 1 20 are predominantly white because black in these subblocks denotes changes from white to black or black to white in the data blocks from which to high frequency subblocks are generated. The changes, which are sometimes called edges, from white to black as well as black to white in Figure 1 25 result in high frequency data values in the HG, HG and GG quadrants as illustrated in Figure 3.

Once the image data has been filtered the desired number of times using the above method, the resulting transformed data values are encoded using a digital code 30 such as the Huffman code in Table 1.

WO 94/23385 PCT/GB94/00677

- 6 -

	Corresponding <pre>Gray-Scale</pre>	Digital <u>Value</u>	Digital <u>Code</u>
		•	
5		•	
5		<u>:</u>	
		5	1000001
		4	100001
		3	10001
		2	1001
10	black	· 1	101
	white	0 .	0
		-1	111
		-2	1101
		-3	11001
15		-4	110001
		- 5	1100001
		•	
		•	

20 Table 1

Because the high frequency components of the original image of Figure 1 are predominantly white as is evident from Figures 3 and 4, the gray-scale white is assigned the single bit 0 in the above digital code. The next most 25 common gray-scale color in the transformed image is black. Accordingly, gray-scale black is assigned the next shortest code of 101. The image of Figure 1 is comprised only of black and white pixels. If the image were to involve other gray-scale shades, then other codes would be 30 used to encode those gray-scale colors, the more predominant gray-scale shades being assigned the relatively shorter codes. The result of the Huffman encoding is that the digital values which predominate in the high frequency components are coded into codes having 35 a few number of bits. Accordingly, the number of bits required to represent the original image data is reduced. The image is therefore said to have been compressed.

Problems occur during compression, however, when the digital filters operate at the boundaries of the data 40 values. For example, when the high pass digital filter generating the high pass component begins generating high pass data values of octave 0 at the left hand side of the original image data, some of the filter inputs required by

the filter do not exist.

Figure 5 illustrates the four data values required by a four coefficient high pass digital filter G in order to generate the first high pass data value G_0 of octave 0. As 5 shown in Figure 5, data values D_1 , D_2 , D_3 and D_4 are required to generate the second high pass data value of octave 0, data value G_1 . In order to generate the first high pass component output data value G_0 , on the other hand, data values D_{-1} , D_0 , D_1 , and D_2 are required. Data 10 value D_{-1} does not, however, exist in the original image data.

Several techniques have been developed in an attempt to solve the problem of the digital filter extending beyond the boundaries of the image data being transformed. 15 In one technique, called zero padding, the nonexistent data values outside the image are simply assumed to be zeros. This may result in discontinuities at the boundary, however, where an object in the image would otherwise have extended beyond the image boundary but 20 where the assumed zeros cause an abrupt truncation of the object at the boundary. In another technique, called circular convolution, the two dimensional multi-octave transform can be expressed in terms of one dimensional finite convolutions. Circular convolution joins the ends 25 of the data together. This introduces a false discontinuity at the join but the problem of data values extending beyond the image boundaries no longer exists. In another technique, called symmetric circular convolution, the image data at each data boundary is 30 mirrored. A signal such as a ramp, for example, will become a peak when it is mirrored. In another technique, called doubly symmetric circular convolution, the data is not only mirrored spatially but the values are also mirrored about the boundary value. This method attempts 35 to maintain continuity of both the signal and its first derivative but requires more computation for the extra mirror because the mirrored values must be pre-calculated

before convolution.

Figure 6 illustrates yet another technique which has been developed to solve the boundary problem. According to this technique, the high and low pass digital filters 5 are moved through the data values in a snake-like pattern in order to eliminate image boundaries in the image data. After the initial one dimensional convolution, the image contains alternating columns of low and high pass information. By snaking through the low pass sub-band 10 before the high pass, only two discontinuities are introduced. This snaking technique, however, requires reversing the digital filter coefficients on alternate rows as the filter moves through the image data. changing of filter coefficients as well as the requirement 15 to change the direction of movement of the digital filters through various blocks of data values makes the snaking technique difficult to implement. Accordingly, an easily implemented method for solving the boundary problem is sought which can be used in data compression and 20 decompression.

Not only does the transformation result in problems at the boundaries of the image data, but the transformation itself typically requires a large number of complex computations and/or data rearrangements. The time 25 required to compress and decompress an image of data values can therefore be significant. Moreover, the cost of associated hardware required to perform the involved computations of the forward transform and the inverse transform may be so high that the transform method cannot 30 be used in cost-sensitive applications. A compression and decompression method is therefore sought that not only successfully handles the boundary problems associated with the forward transform and inverse transform but also is efficiently and inexpensively implementable in hardware 35 and/or software. The computational complexity of the method should therefore be low.

In addition to transformation and encoding, even

further compression is possible. A method known as tree encoding may, for example, be employed. Moreover, a method called quantization can be employed to further compress the data. Tree encoding and quantization are described in various texts and articles including "Image Compression using the 2-D Wavelet Transform" by A.S. Lewis and G. Knowles, published in IEEE Transactions on Image Processing, April 1992. Furthermore, video data which comprises sequences of images can be compressed by taking advantage of the similarities between successive images. Where a portion of successive images does not change from one image to the next, the portion of the first image can be used for the next image, thereby reducing the number of bits necessary to represent the sequence of images.

JPEG (Joint Photographics Experts Group) is an 15 international standard for still-images which typically achieves about a 10:1 compression ratios for monochrome images and 15:1 compression ratios for color images. The JPEG standard employs a combination of a type of Fourier 20 transform, known as the discrete-cosine transform, in combination with quantization and a Huffman-like code. MPEG1 (Motion Picture Experts Group) and MPEG2 are two international video compression standards. MPEG2 is a standard which is still evolving which is targeted for 25 broadcast television. MPEG2 allows the picture quality to be adjusted to allow more television information to be transmitted, e.g., on a given coaxial cable. H.261 is another video standard based on the discrete-cosine transform. H.261 also varies the amount of compression 30 depending on the data rate required.

Compression standards such as JPEG, MPEG1, MPEG2 and H.261 are optimized to minimize the signal to noise ratio of the error between the original and the reconstructed image. Due to this optimization, these methods are very complex. Chips implementing MPEG1, for example, may be costly and require as many as 1.5 million transistors. These methods only partially take advantage of the fact

WO 94/23385 PCT/GB94/00677

- 10 -

that the human visual system is quite insensitive to signal to noise ratio. Accordingly, some of the complexity inherent in these standards is wasted on the human eye. Moreover, because these standards encode by 5 areas of the image, they are not particularly sensitive to edge-type information which is of high importance to the human visual system. In view of these maladaptions of current compression standards to the characteristics of the human visual system, a new compression and 10 decompression method is sought which handles the above-described boundary problem and which takes advantage of the fact that the human visual system is more sensitive to edge information than signal to noise ratio so that the complexity and cost of implementing the method can be 15 reduced.

SUMMARY

A compression and decompression method using wavelet decomposition, frequency based tree encoding, tree based motion encoding, frequency weighted quantization, Huffman 20 encoding, and tree based activity estimation for bit rate control is disclosed. Forward and inverse quasi-perfect reconstruction transforms are used to generate the wavelet decomposition and to reconstruct data values close to the original data values. The forward and inverse quasi25 perfect reconstruction transforms utilize special filters at the boundaries of the data being transformed and/or inverse transformed to solve the above-mentioned boundary problem.

In accordance with some embodiments of the present
inverse, a decompression method uses four coefficient
inverse perfect reconstruction digital filters. The
coefficients of these inverse perfect reconstruction
digital filters require a small number of additions to
implement thereby enabling rapid decompression in software
executing on a general purpose digital computer having a
microprocessor. The method partially inverse transforms a

sub-band decomposition to generate a small low frequency component image. This small image is expanded in one dimension by performing interpolation on the rows of the small image and is expanded in a second dimension by 5 replicating rows of the interpolated small image. Transformed chrominance data values are inverse transformed using inverse perfect reconstruction digital filters having a fewer number of coefficients than the inverse perfect reconstruction digital filters used to 10 inverse transform the corresponding transformed luminance data values. In one embodiment, two coefficient Haar digital filters are used as the inverse perfect reconstruction digital filters which inverse transform transformed chrominance data values. Variable-length 15 tokens are used in the compressed data stream to indicate changes in encoding methods used to encode data values in the compressed data stream.

BRIEF DESCRIPTION OF THE DRAWINGS

Figures 1-4 (Prior Art) are diagrams illustrating a 20 sub-band decomposition of an image.

Figure 5 (Prior Art) is a diagram illustrating a boundary problem associated with the generation of prior art sub-band decompositions.

Figure 6 (Prior Art) is a diagram illustrating a 25 solution to the boundary problem associated with the generation of prior art sub-band decompositions.

Figure 7 is a diagram illustrating a one-dimensional decomposition.

Figures 8 and 9 are diagrams illustrating the 30 separation of an input signal into a high pass component and a low pass component.

Figures 10, 11, 14 and 15 are diagrams illustrating a transformation in accordance with one embodiment of the present invention.

Figures 12 and 13 are diagrams illustrating the operation of high pass and low pass forward transform

- 12 -

PCT/GB94/00677

digital filters in accordance with one embodiment of the present invention.

Figure 16 is a diagram of a two-dimensional matrix of original data values in accordance with one embodiment of the present invention.

Figure 17 is a diagram of the two-dimensional matrix of Figure 16 after one octave of forward transform in accordance with one embodiment of the present invention.

Figure 18 is a diagram of the two-dimensional matrix 10 of Figure 16 after two octaves of forward transform in accordance with one embodiment of the present invention.

Figures 19 and 20 are diagrams illustrating a boundary problem solved in accordance with one embodiment of the present invention.

15 Figure 21 is a diagram illustrating the operation of boundary forward transform digital filters in accordance with one embodiment of the present invention.

Figure 22 is a diagram illustrating the operation of start and end inverse transform digital filters in 20 accordance with one embodiment of the present invention.

Figure 23 is a diagram illustrating a one-dimensional tree structure in accordance one embodiment of the present invention.

Figure 24A-D are diagrams illustrating the recursive 25 filtering of data values to generate a one-dimensional decomposition corresponding with the one-dimensional tree structure of Figure 23.

Figure 25 is a diagram of a two-dimensional tree structure of two-by-two blocks of data values in 30 accordance with one embodiment of the present invention.

Figure 26 is a pictorial representation of the data values of the two-dimension tree structure of Figure 25.

Figures 27-29 are diagrams illustrating a method and apparatus for determining the addresses of data values of 35 a tree structure in accordance with one embodiment of the present invention.

Figure 30 and 31 are diagrams illustrating a

quantization of transformed data values in accordance with one embodiment of the present invention.

Figures 32 and 33 are diagrams illustrating the sensitivity of the human eye to spatial frequency.

Figures 34 is a diagram illustrating the distribution of high pass component data values in a four octave wavelet decomposition of the test image Lenna.

Figure 35 is a diagram illustrating the distribution of data values of the test image Lenna before wavelet 10 transformation.

Figure 36 is a block diagram illustrating a video encoder and a video decoder in accordance with one embodiment of the present invention.

Figure 37 is a diagram illustrating modes of the 15 video encoder and video decoder of Figure 36 and the corresponding token values.

Figure 38 is a diagram illustrating how various flags combine to generate a new mode when the inherited mode is send in accordance with one embodiment of the present 20 invention.

Figures 39-40 are diagrams of a black box on a white background illustrating motion.

Figures 41-43 are one-dimensional tree structures corresponding to the motion of an edge illustrated in 25 Figures 39-40.

Figure 44 is a diagram illustrating variable-length tokens in accordance with one embodiment of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

30 QUASI-PERFECT RECONSTRUCTION FILTERS

The wavelet transform was introduced by Jean Morlet in 1984 to overcome problems encountered in analyzing geological signals. See "Cycle-octave and Related Transforms In Seismic Signal Analysis", Goupillaud,

35 Grossman and Morlet, Geoexploration, vol. 23, 1984. Since then, the wavelet transform has been a new and exciting

method of analyzing signals and has already been applied to a wide range of tasks such as quantum mechanics and signal processing. The wavelet transform has a number of advantages over more traditional Fourier techniques

5 principally used today in the analysis of signals. The wavelet transform and the high and low pass four coefficient quasi-perfect reconstruction filters of the present invention are therefore described by relating them to the windowed Fourier transform.

The windowed Fourier transform is the principle transform used today to analyze the spectral components of a signal. The Fourier transform decomposes a signal under analysis into a set of complex sinusoidal basis functions. The resulting Fourier series can be interpreted as the 15 frequency spectra of the signal. The continuous Fourier transform is defined as follows:

$$F(\omega) = \int_{-\infty}^{\infty} e^{-j2\pi\omega t} f(t) dt \qquad (equ. 1)$$

Where f(t) is the time domain signal under analysis and $F(\omega)$ is the Fourier transform of the signal under 20 analysis. Although many applications require an estimate of the spectral content of an input signal, the above formula is impractical for most systems. In order to calculate the Fourier transform, the input signal f(t) must be defined for all values of time t, whereas in most 25 practical systems, f(t) is only defined over a finite range of time.

Several methods have therefore been devised to transform the finite input signal into an infinite signal so that the Fourier transform can be applied. The 30 windowed Fourier transform is one such solution. The windowed Fourier transform is defined as follows:

$$F_{\mu}(\omega,\tau) = \int_{-\infty}^{\infty} \omega(t-\tau) e^{-j2\pi\omega t} f(t) dt \qquad (equ. 2)$$

Where f(t) is the time domain signal under analysis,

 $F_{w}(\omega,\tau)$ is the windowed Fourier transform of the time domain signal under analysis, and w(t) is the windowing function. The windowing function is usually chosen to be zero outside an interval of finite length. Alternatively, 5 as the spectral content of the input f(t) varies with time, the input signal can be examined by performing the transform at time t using a more local window function. In either case, the output transform is the convolution of the window function and the signal under analysis so that 10 the spectra of the window itself is present in the transform results. Consequently, the windowing function is chosen to minimize this effect. Looking at this technique from another viewpoint, the basis functions of a windowed Fourier transform are not complex sinusoids but 15 rather are windowed complex sinusoids. Dennis Gabor used a real Gaussian function in conjunction with sinusoids of varying frequencies to produce a complete set of basis functions (known as Gabor functions) with which to analyze a signal. For a locality given by the effective width of 20 the Gaussian function, the sinusoidal frequency is varied such that the entire spectrum is covered.

The wavelet transform decomposes a signal into a set of basis functions that can be nearly local in both frequency and time. This is achieved by translating and 25 dilating a function $\Psi(t)$ that has spatial and spectral locality to form a set of basis functions:

$$\sqrt{s}\psi(s(t-u))$$
 (equ. 3)

wherein s and u are real numbers and are the variables of the transform. The function $\Psi(t)$ is called the wavelet.

The continuous wavelet transform of a signal under analysis is defined as follows:

$$W(s,u) = \sqrt{s} \int_{-\infty}^{\infty} \psi \left(s(t-u) \right) f(t) dt \qquad (equ. 4)$$

Where f(t) is the time domain signal under analysis,

WO 94/23385 PCT/GB94/00677

- 16 -

W(s,u) is its wavelet transform, Ψ is the wavelet, s is the positive dilation factor and u is the scaled translation distance. The spatial and spectral locality of the wavelet transform is dependent on the character-5 istics of the wavelet.

Because the signal under analysis in the compression of digitally sampled images has finite length, the discrete counterpart of the continuous wavelet transform is used. The wavelet transform performs a multiresolution decomposition based on a sequence of resolutions often referred to as "octaves". The frequencies of consecutive octaves vary uniformly on a logarithmic frequency scale. This logarithmic scale can be selected so that consecutive octaves differ by a factor of two in frequency. The basis functions are:

$$\{\psi^{j}(x-2^{-j}n)\}\ for\ (j,n)\in\mathbb{Z}^{2}$$
 (equ. 5)

where Z is the set of all integers, $Z^2 = \{(j,n) : j,n \in Z\}$, and $\psi^j(x) = \sqrt{2^j} \psi(2^j x)$.

In a sampled system, a resolution r signifies that 20 the signal under analysis has been sampled at r samples per unit length. A multiresolution analysis studies an input signal at a number of resolutions, which in the case of the present invention is the sequence $r=2^j$ where $j \in Z$. The difference in frequency between consecutive octaves therefore varies by a factor of two.

Stephane Mallat formalized the relationship between wavelet transforms and multiresolution analysis by first defining a multiresolution space sequence $\{V_j\}_{j\in Z}$, where V_j is the set of all possible approximated signals at resolution 2^j . He then showed that an orthonormal basis for V_j can be constructed by $\{\phi^j(x-2^{-j}n)\}_{n\in Z}, \phi(x)$ is called the scaling function where for any $j\in Z$, $\phi^j(x)=\sqrt{2^j}\phi(2^jx)$. He then showed that a signal f(x) can be approximated at a resolution 2^j by the set of samples:

$$S_j = \{\sqrt{2^j} \langle f, \phi_n^j \rangle\}_{n \in \mathbb{Z}}$$
 (equ. 6)

where $\langle f,g \rangle = \int_{-\infty}^{\infty} f(x) \, g(x) \, dx$, where $f,g \in L^2(R)$, the set of square integrable functions on R. This is equivalent to convolving the signal f(x) with the scaling 5 function $\phi^j(-x)$ at a sampling rate of 2^j . However, this representation is highly redundant because $V_j \subset V_{j+1}, j \in \mathbb{Z}$. It would be more efficient to generate a sequence of multiresolution detail signals O_j which represents the difference information between successive resolutions 10 $O_j \oplus V_j = V_{j+1}$ where O_j is orthogonal to V_j . Mallat proved that there exists a function $\Psi(x)$ called the wavelet where:

$$\psi^{j}(x) = \sqrt{2^{j}}\psi(2^{j}x)$$
 (equ. 7)

such that $\{\Psi^{i}(x-2^{-j}n)\}_{mZ}$ is an orthonormal basis of O_{j} and $\{\Psi^{i}(x-2^{-j}n)\}$, $(j,n)\in\mathbb{Z}^{2}$, is an orthonormal basis of $L^{2}(\mathbb{R})$.

15 The detail signal at resolution 2^{j+1} is represented by the set of data values:

$$N_j = \{\sqrt{2^j} < f, \psi_n^j > \}_{n \in \mathbb{Z}}$$
 (equ. 8)

which is equivalent to convolving the signal f(x) with the wavelet $\Psi^{j}(-x)$ at a sampling rate of 2^{j} .

Hence, the original signal f(x) can be completely represented by the sets of data values (S₁, (N₁) J≤j≤-1), where J<O gives the number of octaves. This representation in the form of data values is known as the discrete wavelet decomposition. The S₁ notation used by 25 Mallat refers to recursively low pass filter values of the original signal. S₀ corresponds to the original data values D. S₁ corresponds to the H data values from the low pass filter. N₁ corresponds to the G data values from the high pass filter. S₂ corresponds to the next low pass filtered values from the previous H sub-band. N₂ corresponds to the next high pass filtered values from the previous H sub-band.

If the sampling patterns of the discrete windowed

Fourier transform and the discrete wavelet transform are compared while maintaining the spatial locality of the highest frequency sample for both transforms, then the efficiency of the discrete wavelet decomposition is 5 revealed. The window Fourier transform produces a linear sampling grid, each data value being a constant spatial distance or a constant frequency away from its neighbor. The result is a heavy over-sampling of the lower frequencies. The wavelet transform, in contrast, samples 10 each of its octave wide frequency bands at the minimum rate such that no redundant information is introduced into the discrete wavelet decomposition. The wavelet transform is able to achieve highly local spatial sampling at high frequencies by the use of octave wide frequency bands. At 15 low frequencies, spectral locality takes precedence over spatial locality.

Figure 7 illustrates the spatial and spectral locality of a sequence of sampled data values. The box surrounding a data value represents the spatial and 20 spectral locality of the data value. The regions of Figure 7 are presented for explanation purposes. In reality there is some overlap and aliasing between adjacent data values, the characteristics of which are determined by the particular wavelet function used.

Mallat showed the wavelet transform can be computed with a pyramid technique, where only two filters are used. Using this technique, S_j and N_j are calculated from S_{j+1} , S_j being used as the input for the next octave of decomposition. A low pass filter H:

30
$$h(n) = \frac{1}{\sqrt{2}} \langle \phi_0^{-1}, \phi_n^0 \rangle$$
 (equ. 9)

Mallat showed that S_j can be calculated by convolving from S_{j+1} with H and keeping every other output (i.e. subsampling by a factor of 2).

A method for calculating N_j from S_{j+1} can also be 35 derived. This method involves convolving S_{i+1} with a high

pass filter G and sub-sampling by a factor of 2. The high pass filter G is defined by the following coefficients:

$$g(n) = (-1)^{1-n} h(1-n)$$
 (equ. 10)

The relationship between the H and G filters results in a large saving when the filters are implemented in hardware.

Figures 8 and 9 illustrate that these two filters H and G form a complementary pair that split an input signal into two half band output signals. Both the high and the 10 low pass outputs can be sub-sampled by a factor of two without corrupting the high frequency information because any aliasing introduced by the sub-sampling will be corrected in the reconstruction. There are the same number of filtered data values as there are original image 15 data values.

The particular wavelet which is best in analyzing a signal under analysis is heavily dependent on the characteristics of the signal under analysis. The closer the wavelet resembles the features of the signal, the more 20 efficient the wavelet representation of the signal will In addition, reconstruction errors introduced by quantization resemble the wavelet. Typically, the amount of aliasing varies with spatial support (the number of coefficients of the wavelet filters). Long wavelets can 25 be constructed such that aliasing between adjacent octave bands is minimized. However, the spatial equivalent of aliasing, overlap, increases with filter length. Conversely, short wavelets have little or no overlap spatially but exhibit large amounts of aliasing in the 30 frequency domain. To properly determine the suitability of a wavelet for a particular application, these factors of size and shape must be considered.

To apply the wavelet transform to image processing, the present invention employs a particular wavelet called 35 the four coefficient Daubechies wavelet. Because the four

coefficient Daubechies wavelet has only four coefficients, it is very short. This is well-suited for analyzing important image features such as object edges. Edges by definition are spatially local discontinuities. Edges often consist of a wide spectral range which, when filtered through a high pass filter, give rise to

- filtered through a high pass filter, give rise to relatively larger filtered outputs only when the analysis filter coincides with the edge. When the analysis filter does not coincide with the edge, relatively smaller
- 10 filtered outputs are output by the filter. The shorter the analysis filter used, the more finely the spatial position of the edge is resolved. Longer filters produce more of the relatively larger data values to represent an edge. The shortness of the filter also makes the
- 15 transform calculation relatively inexpensive to implement compared with that of longer filters or image transformations such as the Fourier or discrete cosine transforms. The four coefficient Daubechies wavelet was selected for use only after a careful analysis of both its
- 20 spatial and aliasing characteristics. Longer wavelets such as the six coefficient Daubechies wavelet could, however, also be used if a more complex implementation were acceptable. Short filters such as the two coefficients Haar wavelet could also be used if the 25 attendant high levels of noise were acceptable.

The true coefficients of the four coefficient Daubechies wavelet are:

$$a = \frac{1+\sqrt{3}}{8}$$
, $b = \frac{3+\sqrt{3}}{8}$, $c = \frac{3-\sqrt{3}}{8}$, $d = \frac{-1+\sqrt{3}}{8}$ (equ. 11)

The low pass four coefficient Daubechies digital 30 filter is given by:

$$H\left(\frac{x}{2}\right) = aD(x-1) + bD(x) + cD(x+1) - dD(x+2)$$
 (equ. 12)

The high pass four coefficient Daubechies digital filter is given by:

PCT/GB94/00677

$$G\left(\frac{x}{2}\right) = dD(x-1) + cD(x) - bD(x+1) + aD(x+2)$$
 (equ. 13)

In equations 12 and 13, D(x-1), D(x), D(x+1) and D(x+2) are four consecutive data values. $H\left(\frac{X}{2}\right)$ and $G\left(\frac{X}{2}\right)$ are true perfect reconstruction filters, i.e. the inverse transform 5 perfectly reconstructs the original data. For example, when the filters operate on data values D(1), D(2), D(3) and D(4), outputs H(1) and G(1) are generated. Index x in this case would be 2. Due to the presence of the $\frac{X}{2}$ as the index for the filters H and G, the values of x can 10 only be even integers.

To simplify the computational complexity involved in performing the transformation on real data, the coefficients of the four coefficient Daubechies filter which are non-rational numbers are converted into rational numbers which can be efficiently implemented in software or hardware. Floating point coefficients are not used because performing floating point arithmetic is time consuming and expensive when implemented in software or hardware.

To convert the four Daubechies coefficients for implementation, three relationships of the coefficients a, b, c and d are important. In order for the H filter to have unity gain, the following equation must hold:

$$a + b + c - d = 1$$
 (equ. 14)

25 In order for the G filter to reject all zero frequency components in the input data values, the following equation must hold:

$$a - b + c + d = 0$$
 (equ. 15)

In order for the resulting H and G filters to be able to generate a decomposition which is perfectly reconstructible into the original image data the following equation must hold:

$$ac - bd = 0$$
 (equ. 16)

True four coefficient Daubechies filters satisfy the above three equations 14, 15, and 16. However, when the coefficients of the true low and high pass four 5 coefficient Daubechies filters are converted for implementation, at least one of the three relationships must be broken. In the preferred embodiment, unity gain and the rejection of all zero frequency components are maintained. It is the third relationship of equation 16 10 that is compromised. Perfect reconstruction is compromised because the process of compressing image data itself inherently introduces some noise due to the tree coding and quantization of the present invention. reconstructed data values therefore necessarily involve 15 noise when a real-world image is compressed and then reconstructed. We define filters which satisfy equations 14, and 15 and approximately satisfy equation 16, quasi-perfect reconstruction filters.

Table 2 illustrates a process of converting the 20 coefficients a, b, c and d for implementation.

$$a = \frac{1+\sqrt{3}}{8} = .3415(32) = 10.92 = \frac{11}{32}$$

$$b = \frac{3+\sqrt{3}}{8} = .5915(32) = 18.92 = \frac{19}{32}$$

$$c = \frac{3-\sqrt{3}}{8} = .1585(32) = 5.072 = \frac{5}{32}$$

$$d = \frac{-1+\sqrt{3}}{8} = .0915(32) = 2.928 = \frac{3}{32}$$

Table 2

The true four coefficient Daubechies filter coefficients are listed in the left hand column of Table 2. In the next column to the right, the true coefficients are shown 30 rounded to four places beyond the decimal point. The

rounded coefficients are scaled by a factor of 32 to achieve the values in the next column to the right. From each value in the third column, an integer value is selected. Which integers are selected has a dramatic effect on the complexity of the software or hardware which compresses the image data. The selected integers are divided by 32 so that the scaling by 32 shown in the second column does not change the values of the resulting converted coefficients.

In selecting the integers for the fourth column, the relationship of the three equations 14, 15 and 16 are observed. In the case of a = 11/32, b = 19/32, c = 5/32 and d = 3/32, the relationships a+b+c-d=1 and a-b+c+d=0 both are maintained. Because the converted coefficients in the rightmost column of Table 2 are quite close to the true coefficient values in the leftmost column, the resulting four coefficient filters based on coefficients a, b, c and d allow near perfect reconstruction. On a typical 640 by 480 image, the error between the original and reconstructed data values after forward and then inverse transformation has been experimentally verified to exceed 50 dB.

The resulting high pass four coefficient quasi-Daubechies filter is:

25
$$H(\frac{x}{2}) = \frac{11}{32}D(x-1) + \frac{19}{32}D(x) + \frac{5}{32}D(x+1) - \frac{3}{32}D(x+2)$$
 (equ. 17)

The resulting low pass four coefficient quasi-Daubechies filter is:

$$G(\frac{x}{2}) = \frac{3}{32}D(x-1) + \frac{5}{32}D(x) - \frac{19}{32}D(x+1) + \frac{11}{32}D(x+2)$$
 (equ. 18)

Because the high and low pass four coefficient quasi-30 Daubechies filters satisfy equations 14 and 15 and approximately satisfy equation 16, the high and low pass four coefficient quasi-Daubechies filters are quasiperfect reconstruction filters. WO 94/23385 PCT/GB94/00677

Note that the particular converted coefficients of the quasi-Daubechies filters of equations 17 and 18 result in significant computational simplicity when implementation is either software and/or hardware. 5 Multiplications and divisions by factors of two such as multiplications and divisions by 32 are relatively simple to perform. In either hardware or software, a multiplication by 2 or a division by 2 can be realized by Because the data values being operated on by the 10 digital filter already exist in storage when the filter is implemented in a typical system, the shifting of this data after the data has been read from storage requires little additional computational overhead. Similarly, changing the sign of a quantity involves little additional 15 overhead. In contrast, multiplication and division by numbers that are not a power of 2 require significant overhead to implement in both software and hardware. The selection of the coefficients in equations 17 and 18 allows H(x) and G(x) to be calculated with only additions 20 and shifts. In other words, all multiplications and divisions are performed without multiplying or dividing by . a number which is not a power of 2. Due to the digital filter sequencing through the data values, pipelining techniques can also be employed to reduce the number of 25 adds further by using the sums or differences computed

Moreover, the magnitudes of the inverse transform filter coefficients are the same as those of the transform filter itself. As described further below, only the order 30 and signs of the coefficients are changed. This reduces the effective number of multiplications which must be performed by a factor of two when the same hardware or software implementation is to be used for both the forward and inverse transform. The fact that the signal being 35 analyzed is being sub-sampled reduces the number of additions by a factor of two because summations are required only on the reading of every other sample. The

when the filters were operating on prior data values.

5

effective number of filters is therefore only one to both transform the data into the decomposition and to inverse transform the decomposition back into the image data.

IMAGE COMPRESSION AND DECOMPRESSION USING THE QUASI-PERFECT RECONSTRUCTION TRANSFORM

Color images can be decomposed by treating each Red-Green-Blue (or more usually each Luminance-Chrominance-Chrominance channel) as a separate image. In the case of Luminance-Chrominance (YUV or YIQ) images the 10 chrominance components may already have been sub-sampled. It may be desirable therefore, to transform the chrominance channels through a different number of octaves than the luminance channel. The eye is less sensitive to chrominance at high spatial frequency and therefore these 15 channels can be sub-sampled without loss of perceived quality in the output image. Typically these chrominance channels are sub-sampled by a factor of two in each dimension so that they together take only 50 percent of the bandwidth of the luminance channel. When implementing 20 an image compression technique, the chrominance channels are usually treated the same way as the luminance channel. The compression technique is applied to the three channels independently. This approach is reasonable except in the special cases where very high compression ratios and very 25 high quality output are required. To squeeze the last remaining bits from a compression technique or to achieve more exacting quality criteria, knowledge of how the chrominance rather than luminance values are perceived by the human visual system can be applied to improve the 30 performance of the compression technique by better matching it with the human visual system.

Figure 10 is an illustration of a two dimensional matrix of data values. There are rows of data values extending in the horizontal dimension and there are 35 columns of data values extending in the vertical dimension. Each of the data values may, for example, be

an 8-bit binary number of image pixel information such as the luminance value of a pixel. The data values of Figure 10 represent an image of a black box 100 on a white background 101.

- To transform the data values of the image of Figure 10 in accordance with one aspect of the present invention, a high pass four coefficient quasi-Daubechies digital filter is run across the data values horizontally, row by row, to result in a block 102 of high pass output values G shown in Figure 11. The width of the block 102 of high pass output values in Figure 11 is half the width of the original matrix of data values in Figure 10 because the high pass four coefficient quasi-Daubechies digital filter is moved across the rows of the data values by twos.
- 15 Because only one additional digital filter output is generated for each additional two data values processed by the digital filter, the data values of Figure 10 are said to have been sub-sampled by a factor of two.

Figure 12 illustrates the sub-sampling performed by 20 the high pass digital filter. High pass output G_1 is generated by the high pass digital filter from data values D_1 , D_2 , D_3 and D_4 . The next high pass output generated, output G_2 , is generated by the high pass digital filter from data values D_3 , D_4 , D_5 and D_6 . The high pass digital 25 filter therefore moves two data values to the right for each additional high pass output generated.

A low pass four coefficient quasi-Daubechies digital filter is also run across the data values horizontally, row by row, to generate H block 103 of the low pass 30 outputs shown in Figure 11. This block 103 is generated by sub-sampling the data values of Figure 10 in the same way the block 102 was generated. The H and G notation for the low and high pass filter outputs respectively is used as opposed to the S_j and O_j notation used by Mallat to 35 simplify the description of the two-dimensional wavelet transform.

Figure 13 illustrates the sub-sampling of the low

pass digital filter. Low pass output H₁ is generated by the low pass digital filter from data values D₁, D₂, D₃ and D₄. The next low pass output generated, output H₂, is generated by the low pass digital filter from data values 5 D₃, D₄, D₅ and D₆. The low pass digital filter therefore moves two data values to the right for each additional low pass output generated.

After the high and low pass four coefficient quasi-Daubechies digital filters have generated blocks 102 and 10 103, the high and low pass four coefficient quasi-Daubechies digital filters are run down the columns of blocks 102 and 103. The values in blocks 102 and 103 are therefore sub-sampled again. The high pass four coefficient quasi-Daubechies digital filter generates 15 blocks 104 and 105. The low pass four coefficient quasi-Daubechies digital filter generates blocks 106 and 107. The resulting four blocks 104-107 are shown in Figure 14. Block 106 is the low frequency component of the original image data. Blocks 107, 104 and 105 comprise the high 20 frequency component of the original image data. Block 106 is denoted block HH. Block 107 is denoted block GH. Block 104 is denoted block HG. Block 105 is denoted block GG.

This process of running the high and low pass four

25 coefficient quasi-Daubechies digital filters across data
values both horizontally and vertically to decompose data
values into high and low frequency components is then
repeated using the data values of the HH block 106 as
input data values. The result is shown in Figure 15.

30 Block 108 is the low frequency component and is denoted
block HHHH. Blocks 109, 110 and 111 comprise octave 1 of
the high frequency component and are denoted HHHG, HHGH,
HHGG, respectively. Blocks HG, GH and GG comprise octave
0 of the high frequency component.

Although this recursive decomposition process is only repeated twice to produce high pass component octaves 0 and 1 in the example illustrated in connection with

Figures 10-15, other numbers of recursive decomposition steps are possible. Recursively decomposing the original data values into octaves 0, 1, 2 and 3 has been found to result in satisfactory results for most still image data and recursively decomposing the original data into octaves 0, 1, and 2 has been found to result in satisfactory results for most video image data.

Moreover, the horizontal and subsequent vertical operation of the high and low pass filters can also be 10 reversed. The horizontal and subsequent vertical sequence is explained in connection with this example merely for instructional purposes. The filters can be moved in the vertical direction and then in the horizontal direction. Alternatively, other sequences and dimensions of moving 15 the digital filters through the data values to be processed is possible.

It is also to be understood that if the original image data values are initially arrayed in a two dimensional block as shown in Figure 10, then the 20 processing of the original image data values by the high and low pass filters would not necessarily result in the HH values being located all in an upper right hand quadrant as is shown in Figure 14. To the contrary, depending on where the generated HH values are written, 25 the HH data values can be spread throughout a block. The locations of the HH values are, however, determinable. The HH values are merely illustrated in Figure 14 as being located all in the upper lefthand quadrant for ease of illustration and explanation.

Figure 16 is an illustration showing one possible twelve-by-twelve organization of original image data values in a two dimensional array. Figure 16 corresponds with Figure 10. The location in the array of each data value is determined by a row number and column number. A row number and column number of a data value may, for example, correspond with a row address and column address in an addressed storage medium. This addressed storage

medium may, for example, be a semiconductor memory, a magnetic storage medium, or an optical storage medium. The row and column may, for example, also correspond with a pixel location including a location of a pixel on a 5 cathode-ray tube or on a flat panel display.

Figure 17 is an illustration showing the state of the two dimensional array after a one octave decomposition. The HH low frequency components are dispersed throughout the two dimensional array as are the HG values, the GH 10 values, and the GG values. The subscripts attached to the various data values in Figure 17 denote the row and column location of the particular data value as represented in the arrangement illustrated in Figure 14. HH₀₀, HH₀₁, HH₀₂, HH₀₃, HH₀₄ and HH₀₅, for example, are six data values which 15 correspond with the top row of data values in HH block 106 of Figure 14. HH₀₀, HH₁₀, HH₂₀, HH₃₀, HH₄₀ and HH₃₀, for example, are six data values which correspond with the leftmost column of data values in HH block 106 of Figure 14.

20 When the high and the low pass forward transform digital filters operate on the four data values D_{01} , D_{02} , D_{03} and D_{os} of Figure 16, the output of the low pass forward transform digital filter is written to location row 0 column 2 and the output of the high pass forward transform 25 digital filter is written to location row 0 column 3. Next, the high and low pass forward transform digital filters are moved two locations to the right to operate on the data values D_{00} , D_{04} , D_{05} and D_{06} . The outputs of the low and high pass forward transform digital filters are 30 written to locations row 0 column 4 and row 0 column 5, respectively. Accordingly, the outputs of the low and high frequency forward transform digital filters are output from the filters to form an interleaved sequence of low and high frequency component data values which 35 overwrite the rows of data values in the two dimensional array.

Similarly, when the low and high pass forward

transform digital filters operate on the four data values at locations column 0, rows 1 through 4, the output of the low pass forward transform digital filter is written to location column 0 row 2. The output of the high pass 5 forward transform digital filter is written to location column 0 row 3. Next the low and high pass forward transform digital filters are moved two locations downward to operate on the data values at locations column 0, rows 3 through 6. The outputs of the low and high pass forward 10 transform digital filters are written to locations column 0 row 4 and column 0 row 5, respectively. Again, the outputs of the low and high pass forward transform digital filters are output from the filters in an interleaved fashion to overwrite the columns of the two dimensional 15 array.

Figure 18 is an illustration showing the state of the two dimensional array after a second octave decomposition. The HHHH low frequency components corresponding which block 108 of Figure 15 as well as the octave 1 high 20 frequency components HHGH, HHHG and HHGG are dispersed throughout the two dimensional array. When the HH values $HH_{01},\ HH_{02},\ HH_{03}$ and HH_{04} of Figure 17 are processed by the low and high pass forward transform digital filters, the outputs are written to locations row 0 column 4 and row 0 25 column 6, respectively. Similarly, when the values at locations column 0, rows 2, 4, 6 and 8 are processed by the low and high pass forward transform digital filters, the results are written to locations column 0 row 4 and column 0 row 6, respectively. The data values in Figure 30 18 are referred to as transformed data values. transformed data values are said to comprise the decomposition of the original image values.

This method of reading data values, transforming the data values, and writing back the output of the filters is 35 easily expanded to a two dimensional array of a very large size. Only a relatively small number of locations is shown in the two dimensional array of Figures 10-18 for

WO 94/23385

- 31 -

ease of explanation and clarity of illustration.

The transformed data values are reconverted back into image data values substantially equal to the original image data by carrying out a reverse process. 5 reverse process is called the inverse transform. Due to the interleaved nature of the decomposition data in Figure 18, the two digital filters used to perform the inverse transform are called interleaved inverse transform digital filters. Odd data values are determined by an odd 10 interleaved inverse digital filter O. Even data values are determined by the even interleaved inverse transform digital filter E.

The odd and even interleaved inverse digital filters can be determined from the low and high pass forward 15 transform digital filters used in the forward transform because the coefficients of the odd interleaved inverse transform digital filters are related to the coefficients of the low and high pass forward transform filters. To determine the coefficients of the odd and even interleaved 20 inverse transform digital filters, the coefficients of the low and high pass forward transform digital filters are reversed. Where the first, second, third and fourth coefficients of the low pass forward transform digital filter H of equation 17 are denoted a, b, c and -d, the 25 first, second, third and fourth coefficients of a reversed filter H* are denoted -d, c, b and a. Similarly, where the first, second, third and fourth coefficients of the high pass forward transform digital filter G of equation 18 are denoted d, c, -b and a, the first, second, third 30 and fourth coefficients of a reverse filter G* are denoted a, -b, c and d.

The first through the fourth coefficients of the even interleaved inverse transform digital filter E are the first coefficient of H*, the first coefficient of G*, the 35 third coefficient of H*, and the third coefficient of G*. The coefficients of the even interleaved inverse transform digital filter E therefore are -d, a, b and c.

case of the low and high pass four coefficient quasi-Daubechies filters used in the transform where $a = \frac{11}{32}$, $b = \frac{19}{32}$, $c=\frac{5}{32}$ and $d=\frac{3}{32}$, the even interleaved inverse transform digital filter is:

$$5 \frac{D(2x)}{2} = -\frac{3}{32}H(x-1) + \frac{11}{32}G(x-1) + \frac{19}{32}H(x) + \frac{5}{32}G(x) \text{ (equ. 19)}$$

where H(x-1), G(x-1), H(x) and G(x) are transformed data values of a decomposition to be inverse transformed.

The first through the fourth coefficients of the odd . interleaved inverse transform digital filter O are the 10 second coefficient of H*, the second coefficient of G*, the fourth coefficient of H*, and the fourth coefficient of G*. The coefficients of the odd interleaved inverse transform digital filter O therefore are c, -b, a and d. In the case of the low and high pass four coefficient 15 quasi-Daubechies filters used in the transform where $a=\frac{11}{32}$, $b=\frac{19}{32}$, $c=\frac{5}{32}$ and $d=\frac{3}{32}$, the odd interleaved inverse transform digital filter is:

$$\frac{D(2x-1)}{2} = \frac{5}{32}H(x-1) - \frac{19}{32}G(x-1) + \frac{11}{32}H(x) + \frac{3}{32}G(x) \text{ (equ. 20)}$$

where H(x-1), G(x-1), H(x) and G(x) are data values of a 20 decomposition to be inverse transformed. To inverse transform the transformed data values of

Figure 18 into the data values of Figure 17, the HHHG, HHGG, HHGH and data values are inverse transformed with the HHHH data values to create the HH data values of 25 Figure 17. This process corresponds with the inverse transformation of HHHG block 109, HHGH block 110, HHGG block 111, and HHHH block 108 of Figure 15 back into the HH data values of block 106 of Figure 14. The HG, GH and GG data values of Figure 18 are therefore not processed by 30 the odd and even interleaved inverse transform digital

filters in this step of the inverse transform.

In Figure 18, the odd interleaved inverse transform digital filter processes the values in locations column 0, rows 0, 2, 4 and 6 to generate the odd data value at location column 0 row 2. The even interleaved inverse 5 transform digital filter data also processes the values in the same locations to generate the even data value at location column 0 row 4. The odd and even interleaved inverse transform digital filters then process the values in locations column 0, rows 4, 6, 8 and A to generate the 10 values at locations column 0 row 6 and column 0 row 8, respectively. Each of the six columns 0, 2, 6, 4, 8, and A of the values of Figure 18 are processed by the odd and even interleaved inverse transform digital filters in accordance with this process.

- The various locations are then processed again by the odd and even interleaved inverse transform digital filters, this time in the horizontal direction. The odd and even interleaved inverse transform digital filters process the values at locations row 0 columns 0, 2, 4 and
- 20 6 to generate the values at locations row 0 column 2 and row 0 column 4, respectively. The odd and even interleaved inverse transform digital digital filters process the values at locations row 0 columns 4, 6, 8 and A to generate the values at locations row 0 column 6 and
- 25 row 0 column 8, respectively. Each of the six rows 0, 2, 4 and 8 and of values are processed by the even and odd interleaved inverse transform digital filters in accordance with this process. The result is the reconstruction shown in Figure 17.
- The even and odd interleaved inverse transform digital filters then process the values shown in Figure 17 into the data values shown in Figure 16. This inverse transformation corresponds with the transformation of the HH block 106, the HG bock 104, the GH block 107 and the GG block 105 of Figure 14 into the single block of data value of Figure 10. The resulting reconstructed data values of Figure 16 are substantially equal to the original image

WO 94/23385 PCT/GB94/00677

- 34 -

data values.

Note, however, that in the forward transform of the data values of Figure 16 into the data values of Figure 17 that the low and high pass four coefficient quasi
5 Daubechies digital filters cannot generate all the data values of Figure 17 due to the digital filters requiring data values which are not in the twelve by twelve matrix of data values of Figure 16. These additional data values are said to be beyond the "boundary" of the data values to 10 be transformed.

Figure 19 illustrates the high pass four coefficient quasi-Daubechies digital filter operating over the boundary to generate the G₀ data value. generate the G_0 data value in the same fashion that the 15 other high frequency G data values are generated, the high pass digital filter would require data values $D_{.i}$, D_0 , D_1 and D_2 as inputs. Data value $D_{.1}$, however, does not exist. Similarly, Figure 20 illustrates the low pass four coefficient quasi-Daubechies digital filter operating over 20 the boundary to generate the H_0 data value. In order to generate the Ho data value in the same fashion that the other low frequency H data values are generated, the low pass digital filter would require data values D.1, D0, D1 and D_2 as inputs. Data value $D_{.1}$, however, does not exist. The present invention solves this boundary problem by 25 using additional quasi-Daubechies digital filters to generate the data values adjacent the boundary that would otherwise require the use of data values outside the boundary. There is a high pass "start" quasi-Daubechies 30 forward transform digital filter G, which is used to generate the first high pass output G_0 . There is a low pass "start" quasi-Daubechies forward transform digital filter H, which is used to generate the first low pass output H_0 . These start quasi-Daubechies forward transform 35 digital filters are three coefficient filters rather than four coefficient filters and therefore require only three data values in order to generate an output. This allows

the start quasi-Daubechies forward transform digital filters to operate at the boundary and to generate the first forward transform data values without extending over the boundary.

Figure 21 illustrates the low and high pass start quasi-Daubechies forward transform digital filters operating at the starting boundary of image data values Do through DB. The three coefficient low and high pass start quasi-Daubechies forward transform digital filters operate on data values Do, D1 and D2 to generate outputs Ho and Go, respectively. H1, H2, H3 and H4, on the other hand, are generated by the low pass four coefficient quasi-Daubechies forward transform digital filter and G1, G2, G3 and G4 are generated by the high pass four coefficient quasi-Daubechies forward transform digital filter.

A similar boundary problem is encountered at the end of the data values such as at the end of the data values of a row or a column of a two-dimensional array. If the low and high pass four coefficient quasi-Daubechies 20 filters G and H are used at the boundary in the same fashion that they are in the middle of the data values, then the four coefficient quasi-Daubechies forward transform digital filters would have to extend over the end boundary to generate the last low and high pass outputs, respectively.

The present invention solves this boundary problem by using additional quasi-Daubechies forward transform digital filters in order to generate the transformed data values adjacent the end boundary that would otherwise 30 require the use of data outside the boundary. There is a low pass "end" quasi-Daubechies forward transform digital filter H, which is used to generate the last low pass output. There is a high pass "end" quasi-Daubechies forward transform digital filter G, which is used to 35 generate the last high pass output. These two end quasi-Daubechies forward transform digital filters are three coefficient filters rather than four coefficient filters

and therefore require only three data values in order to generate an output. This allows the end quasi-Daubechies forward transform digital filters to operate at the boundary and to generate the last transform data values without extending over the boundary.

Figure 21 illustrates two low and high pass end quasi-Daubechies forward transform digital filters operating at the end boundary of the image data. These three coefficient low and high pass end quasi-Daubechies forward transform digital filters operate on data values D₉, D_A and D_B to generate outputs H₅ and G₅, respectively. This process of using the appropriate start or end low or high pass filter is used in performing the transformation at the beginning and at the end of each row and column of the data values to be transformed.

The form of the low pass start quasi-Daubechies forward transform digital filter H, is determined by selecting a value of a hypothetical data value D, which would be outside the boundary and then determining the 20 value of the four coefficient low pass quasi-Daubechies forward transform filter if that four coefficient forward transform filter were to extend beyond the boundary to the hypothetical data value in such a way as would be necessary to generate the first low pass output Ho. 25 hypothetical data value D., outside the boundary can be chosen to have one of multiple different values. In some embodiments, the hypothetical data value D, has a value equal to the data value Do at the boundary. In some embodiments, the hypothetical data value D, is set to zero 30 regardless of the data value Do. The three coefficient low pass start quasi-Daubechies forward transform digital filter H, therefore has the form:

$$H_0 = K1 + bD_0 + cD_1 - dD_2$$
 (equ. 21)

where K1 is equal to the product aD_1 , where D_0 is the first 35 data value at the start boundary at the start of a

sequence of data values, and where a, b, c and d are the four coefficients of the four coefficient low pass quasi-Daubechies forward transform digital filter. If, for example, hypothetical data value D_1 is chosen to be equal to the data value D_0 adjacent but within the boundary, then $K1=aD_0$ where a=11/32 and D_0 is the data value adjacent the boundary, equation 21 then becomes:

$$H_0 = (a+b)D_0 + cD_1 - dD_2$$
 (equ. 22)

The form of the high pass start quasi-Daubechies 10 forward transform digital filter G, is determined by the same process using the same hypothetical data value D.1. The high pass start quasi-Daubechies forward transform digital filter G, therefore has the form:

$$G_0 = K2 + cD_0 - bD_1 + aD_2$$
 (equ. 23)

15 where K2 is equal to the product dD₁, where D₀ is the first data value at the boundary at the start of a sequence of data values, and where a, b, c and d are the four coefficients of the four coefficient high pass quasi-Daubechies forward transform digital filter. If
20 hypothetical data value D₁ is chosen to be equal to D₀, then equation 23 becomes:

$$G_0 = (d + c)D_0 - bD_1 + aD_2$$
 (equ. 24)

The form of the low pass end quasi-Daubechies forward transform digital filter H, is determined in a similar way 25 to the way the low pass start quasi-Daubechies forward transform digital filter is determined. A value of a data value D_C is selected which would be outside the boundary. The value of the four coefficient low pass quasi-Daubechies forward transform digital filter is then 30 determined as if that four coefficient filter were to extend beyond the boundary to data value D_C in such a way

as to generate the last low pass output H_3 . The three coefficient low pass end quasi-Daubechies forward transform digital filter therefore has the form:

$$H_5 = aD_9 + bD_A + cD_B - K3$$
 (equ. 25)

5 where K3 is equal to the product dD_C, where D_B is the last data value of a sequence of data values to be transformed, and where a, b, c and d are the four coefficients of the four coefficient low pass quasi-Daubechies filter. D_B is the last data value in the particular sequence of data values of this example and is adjacent the end boundary. In the case where the hypothetical data value D_c is chosen to be equal to the data value D_B adjacent but within the end boundary, then K3=dD_B and equation 25 becomes:

$$H_5 = aD_9 + bD_A + (c-d)D_B$$
 (equ. 26)

The form of the high pass end quasi-Daubechies forward transform digital filter G_c is determined by the same process using the same data value D_C . The three coefficient high pass end quasi-Daubechies forward transform digital filter therefore has the form:

$$G_{5} = dD_{9} + cD_{A} - bD_{B} + K4 \qquad (equ. 27)$$

where K4 is equal to the product aD_c , where D_B is the last data value in this particular sequence of data values to be transformed, and where a, b, c and d are the four coefficients of the four coefficient high pass quasi-Daubechies forward transform digital filter. D_B is adjacent the end boundary. If hypothetical data value D_C is chosen to be equal to D_B , then equation 27 becomes:

$$G_5 = dD_9 + cD_A + (-b+a)D_B$$
 (equ. 28)

It is to be understood that the specific low and high

WO 94/23385 PCT/GB94/00677

- 39 -

pass end quasi-Daubechies forward transform digital filters are given above for the case of data values Do through D_B of Figure 21 and are presented merely to illustrate one way in which the start and end digital 5 filters may be determined. In the event quasi-Daubechies filters are not used for the low and high pass forward transform digital filters, the same process of selecting a hypothetical data value or values outside the boundary and then determining the value of a filter as if the filter 10 were to extend beyond the boundary can be used. embodiments, multiple hypothetical data values may be selected which would all be required by the digital filters operating on the inside area of the data values in order to produce an output at the boundary. This boundary 15 technique is therefore extendable to various types of digital filters and to digital filters having numbers of coefficients other than four.

As revealed by Figure 22, not only does the forward transformation of data values at the boundary involve a 20 boundary problem, but the inverse transformation of the transformed data values back into original image data values also involves a boundary problem. In the present example where four coefficient quasi-Daubechies filters are used to forward transform non-boundary data values, 25 the inverse transform involves an odd inverse transform digital filter as well as an even inverse transform digital filter. Each of the odd and even filters has four coefficients. The even and odd reconstruction filters alternatingly generate a sequence of inverse transformed 30 data values.

In Figure 22, the data values to be transformed are denoted H₀, G₀ ... H₄, G₄, H₅, G₅. Where the forward transform processes the rows first and then the columns, the inverse transform processes the columns first and then 35 the rows. Figure 22 therefore shows a column of transferred data values being processed in a first step of the inverse transform. Both the forward and the inverse

transforms in the described example, however, process the columns in a downward direction and process the rows in a left-right direction.

In Figure 22, the inverse transformed data values 5 reconstructed by the inverse transform digital filters are denoted D_0 , D_1 , D_2 , D_3 ... D_8 . The odd inverse transform digital filter outputs are shown on the left and the even inverse transform digital filter outputs are shown on the right.

At the beginning of the sequence of data values Ho, 10 G_0 , H_1 , G_1 ... H_5 and G_5 to be inverse transformed, the four coefficient odd and even inverse transform digital filters determine the values of reconstructed data values D, and D, using values H_0 , G_0 , H_1 and G_1 , respectively. Reconstructed 15 data value Do, however, cannot be reconstructed from the four coefficient even inverse transform digital filter without the four coefficient even inverse transform digital filter extending beyond the boundary. If the four coefficient even inverse transform filter were to be 20 shifted two data values upward so that it could generate data value Do, then the even four coefficient inverse transform digital filter would require two additional data values to be transformed, data values G, and H, . Ho is, however, the first data value within the boundary and is 25 located adjacent the boundary.

To avoid the even four coefficient inverse transform digital filter extending beyond the boundary, a two coefficient inverse transform digital filter is used:

$$D_0 = 4[(b-a)H_0 + (c-d)G_0]$$
 (equ. 29)

30 in the case where $K1 = aD_0$ and $K2 = dD_0$. D_0 is the first data value and H_0 is the data value to be inverse transformed adjacent the start boundary. This even start inverse transform digital filter has the form of the four coefficient even inverse transform digital filter except 35 that the G_1 data value outside the boundary is chosen to

be equal to H_0 , and the H_{-1} data value outside the boundary is chosen to be equal to G_0 . The even start invere transform digital filter therefore determines D_0 as a function of only H_0 and G_0 rather than as a function of H_{-1} , G_{-1} , $G_{$

Similarly, a two coefficient odd end inverse transform digital filter is used to avoid the four coefficient odd inverse transform digital filter from extending beyond the end boundary at the other boundary of 10 a sequence of data values to be inverse transformed. The two coefficient odd end inverse transform digital filter used is:

$$D_B = 4[(c+d)H_5 - (a+b)G_5]$$
 (equ. 30)

in the case where K4 = aD_B and K3 = dD_B. D_B is the data

15 value to be determined and G₅ is the data value to be
inverse transformed adjacent the end boundary. This odd
end inverse transform digital filter has the form of the
four coefficient odd inverse transform digital filter
except that the H₆ data value outside the boundary is

20 chosen to be equal to G₅ and the G₆ data value outside the
boundary is chosen to be equal to H₅. The odd end inverse
transform digital filter therefore determines D_B as a
function of only H₅ and G₅ rather than as a function of H₅,
G₅, H₆ and G₆.

It is to be understood that the particular even start and odd end inverse transform digital filters used in this embodiment are presented for illustrative purposes only. Where there is a different number of data values to be inverse transformed in a sequence of data values, an even one inverse transform digital filter may be used at the boundary rather than the odd end inverse transform digital filter. The even end inverse transform digital filter is an even inverse transform digital filter modified in accordance with the above process to have fewer coefficients than the even inverse transform digital

filter operating on the inner data values. Where filters other than quasi-Daubechies inverse transform digital filters are used, start and end inverse transform digital filters can be generated from the actual even and odd inverse transform digital filters used to inverse transform data values which are not adjacent to a boundary. In the inverse transform, the start inverse transform digital filter processes the start of the transformed data values at the start boundary, then the four coefficient inverse transform digital filters process the non-boundary transformed data values, and then the end inverse transform digital filter processes the end of the transformed data values.

The true Daubechies filter coefficients a, b, c and d
15 fulfil some simple relationships which show that the
inverse transform digital filters correctly reconstruct
non-boundary original image data values.

$$a+c = \frac{1}{2}$$
, $b-d = \frac{1}{2}$, $c+d = \frac{1}{4}$, $b-a = \frac{1}{4}$ (equ. 31)

and the second order equations:

20 ac-bd = 0,
$$a^2+b^2+c^2+d^2=\frac{1}{2}$$
 (equ. 32)

Take two consecutive H,G pairs:

$$H\left(\frac{x}{2}\right) = aD(x-1)+bD(x)+cD(x+1)-dD(x+2)$$
 (equ. 33)

$$G\left(\frac{x}{2}\right) = dD(x-1)+cD(x)-bD(x+1)+aD(x+2)$$
 (equ. 34)

$$H\left(\frac{x}{2}+1\right) = aD(x+1)+bD(x+2)+cD(x+3)-dD(x+4)$$
 (equ. 35)

25
$$G\left(\frac{x}{2}+1\right) = dD(x+1)+cD(x+2)-bD(x+3)+aD(x+4)$$
 (equ. 36)

Multiplying Equations 33 to 36 using the inverse transform digital filters gives:

$$cH(\frac{x}{2}) = acD(x-1)+bcD(x)+c^2D(x+1)-cdD(x+2)$$
 (equ. 37)

$$-bG\left(\frac{x}{2}\right) = -bdD(x-1) - bcD(x) + b^2D(x+1) - abD(x+2)$$
 (equ. 38)

$$aH(\frac{x}{2}+1) = a^2D(x+1)+abD(x+2)+acD(x+3)-adD(x+4)$$
 (equ. 39)

$$dG\left(\frac{x}{2}+1\right) = d^{2}D(x+1)+cdD(x+2)-bdD(x+3)+adD(x+4)$$
 (equ. 40)

$$5 - dH\left(\frac{x}{2}\right) = -adD(x-1) - bdD(x) - cdD(x+1) + d^2D(x+2)$$
 (equ. 41)

$$aG(\frac{x}{2}) = adD(x-1) + acD(x) - abD(x+1) + a^2D(x+2)$$
 (equ. 42)

$$bH\left(\frac{x}{2}+1\right) = abD(x+1)+b^2D(x+2)+bcD(x+3)-bdD(x+4) \qquad (equ. 43)$$

$$cG(\frac{x}{2}+1) = cdD(x+1)+c^2D(x+2)-bcD(x+3)+acD(x+4)$$
 (equ. 44)

Summing equations 37-40 and 41-44 yields:

10
$$cH(\frac{x}{2}) - bG(\frac{x}{2}) + aH(\frac{x}{2}+1) + dG(\frac{x}{2}+1) =$$

$$(ac-bd)D(x-1) + (a^2+b^2+c^2+d^2)D(x+1) + (ac-bd)D(x+3) = D(x+1)/2$$
(equ. 45)

$$-dH\left(\frac{x}{2}\right) + aG\left(\frac{x}{2}\right) + bH\left(\frac{x}{2}+1\right) + cG\left(\frac{x}{2}+1\right) = (ac-bd)D(x) + (a^2+b^2+c^2+d^2)D(x+2) + (ac-bd)D(x+4) = D(x+2)/2$$
15

(equ. 46)

Using the coefficients of the four coefficient true Daubechies filter, the relationships of equations 31 and 32 hold. Equations 45 and 46 therefore show that with a one bit shift at the output, the original sequence of data 20 values is reconstructed.

Similarly, that the even start reconstruction filter of equation 29 and the odd end reconstruction filter of equation 30 correctly reconstruct the original image data adjacent the boundaries is shown as follows.

For the even start filter, with the choice of $K1 = aD_0$ and $K2 = dD_0$ in equations 29 and 30, we have:

WO 94/23385

$$H_0 = (a+b)D_0 + cD_1 - dD_2$$
 (equ. 47)

$$G_0 = (c+d)D_0 - bD_1 + aD_2$$
 (equ. 48)

so

$$bH_0 = b(a+b)D_0 + cbD_1 - dbD_2$$
 (equ. 49)

5
$$cG_0 = c(c+d)D_0 - cbD_1 + acD_2$$
 (equ. 50)

$$aH_0 = a(a+b)D_0 + acD_1 - adD_2$$
 (equ. 51)

$$dG_0 = d(c+d)D_0 - dbD_1 + adD_2$$
 (equ. 51')

and hence: from equation 29:

$$bH_0 + cG_0 - aH_0 - dG_0 = (b^2-a^2+c^2-d^2)D_0 = \frac{D_0}{A} (equ. 52)$$

For the odd end filter, with the choice of $K_3 = dD_8$ and $K_4 = aD_8$, we have:

$$H_5 = aD_9 + bD_A + (c-d)D_B$$
 (equ. 53)

$$G_5 = dD_9 + cD_A + (a-b)D_B$$
 (equ. 54)

$$cH_5 = acD_9 + bcD_A + c(c-d)D_B$$
 (equ. 55)

15
$$-bG_5 = -bdD_9 - bcD_A - b(a-b)D_B$$
 (equ. 56)

$$dH_5 = daD_9 + bdD_A + d(c-d)D_B \qquad (equ. 57)$$

$$-aG_5 = -adD_9 - caD_A - a(a-b)D_B$$
 (equ. 58)

and hence from equation 30:

$$(c+d)H_5 - (a+b)G_5 = (c^2-d^2+b^2-a^2)D_8 = \frac{D_8}{4}$$
 (equ. 59)

This reveals that the start and end boundary inverse transform digital filters can reconstruct the boundary data values of the original image when low pass and high pass start and end digital filters are used in the forward transform.

TREE ENCODING AND DECODING

As described above, performing the forward quasiperfect inverse transform does not reduce the number of
data values carrying the image information. Accordingly,
10 the decomposed data values are encoded such that not all
of the data values need be stored or transmitted. The
present invention takes advantage of characteristics of
the Human Visual System to encode more visually important
information with a relatively larger number of bits while
15 encoding less visually important information with a
relatively smaller number of bits.

By applying the forward quasi-perfect inverse transform to a two-dimensional array of image data values, a number of sub-band images of varying dimensions and 20 spectral contents is obtained. If traditional sub-band coding were used, then the sub-band images would be encoded separately without reference to each other except perhaps for a weighting factor for each band. This traditional sub-band encoding method is the most readily-25 recognized encoding method because only the spectral response is accurately localized in each band.

In accordance with the present invention, however, a finite support wavelet is used in the analysis of an image, so that the sub-bands of the decomposition include 30 spatially local information which indicate the spatial locations in which the frequency band occurs. Whereas most sub-band encoding methods use long filters in order to achieve superior frequency separation and maximal stop band rejection, the filter used in the present invention 35 has compromised frequency characteristics in order to maintain good spatial locality.

Images can be thought of as comprising three components: background intensities, edges and textures. The forward quasi-perfect inverse transform separates the background intensities (the low pass luminance and 5 chrominance bands) from the edge and texture information contained in the high frequency bands. Ideally, enough bandwidth would be available to encode both the edges and the textures so that the image would reconstruct perfectly. The compression due to the encoding would then 10 be entirely due to removal of redundancy within the picture. If, however, the compressed data is to be transmitted and/or stored at low data transmission rates, some visual information of complex images must be lost. Because edges are a visually important image feature, the 15 encoding method of the present invention locates and encodes information about edges or edge-like features for transmission or storage and places less importance on encoding textural information.

There are no exact definitions of what constitutes an 20 edge and what constitutes texture. The present invention uses a definition of an edge that includes many types of textures. An edge or an edge-like feature is defined as a spatially local phenomenon giving rise to a sharp discontinuity in intensity, the edge or edge-like feature 25 having non-zero spectral components over a range of frequencies. Accordingly, the present invention uses a frequency decomposition which incorporates spatial locality and which is invertible. The wavelet transform realized with quasi-perfect inverse transform digital 30 filters meets these requirements.

Because an edge has non-zero components over a range of frequencies of the decomposition in the same locality, an edge can be located by searching through the wavelet decomposition for non-zero data values that represent sedges. The method begins searching for edges by examining the low frequency sub-bands of the decomposition. These bands have only a small number of data values because of

- .-... - ---

the subsampling used in the wavelet transform and because the spatial support of each low frequency data value is large. After a quick search of the lowest frequency subbands, the positions of potential edges are determined.

- 5 Once the locations of the edges are determined in the lowest frequency sub-bands, these locations can be examined at a higher frequency resolutions to confirm that the edges exist and to more accurately determine their spatial locations.
- Figure 23 illustrates an example of a one-dimensional binary search. There are three binary trees arranged from left to right in the decomposition of Figure 23. There are three octaves, octaves 0, 1 and 2, of decomposed data values in Figure 23. The low pass component is not
- 15 considered to be an octave of the decomposition because most of the edge information has been filtered out.

 Figures 24A-24D illustrate the forward transformation of a one-dimensional sequence of data values D into a sequence of transformed data values such as the tree structure of
- 20 Figure 23. The data values of the sequence of Figure 24A are filtered into low and high frequency components H and G of Figure 24B. The low frequency component of Figure 24B is then filtered into low and high frequency components HH and HG of Figure 24C. The low frequency
- 25 component HH of Figure 24C is then filtered into low and high frequency components HHH and HHG. The transformed data values of HHH block 240 of Figure 24D correspond with the low frequency component data values A, G and M of Figure 23. The transformed data values of HHG block 241
- 30 of Figure 24D correspond with the octave 2 data values B, H and N of Figure 23. The transformed data values of HG block 242 of Figure 24D correspond with the octave 1 data values of Figure 23. Similarly, the transformed data values of G block 243 correspond with the octave 0 data
- 35 values of Figure 23. Although only three trees are shown in Figure 23, the number of HHH data values in block 240 can be large and the size of the tree structure of Figure

number.

23 can extend in the horizontal dimension in a corresponding manner.

The encoding of a one dimensional wavelet decomposition such as the decomposition of Figure 23 is 5 performed in similar fashion to a binary tree search. The spatial support of a given data value in a given frequency band is the same as two data values in the octave above it in frequency. Thus the wavelet decomposition is visualized as an array of binary trees such as is 10 illustrated in Figure 23, each tree representing a spatial locality. The greater the number of transform octaves, the higher the crees extend upward and the fewer their

As illustrated in Figure 23, each of the data values of the decomposition represents a feature which is either "interesting" to the human visual system, or it represents a feature that is "non-interesting" to the human visual system. A data value representing an edge of an object in an image or an edge-like feature is an example of an

- 20 "interesting" data value. The encoding method is a depth first search, which starts at the trunk of a tree, ascends up the branches of the tree that are interesting, and terminates at the non-interesting branches. After all the branches of a tree have been ascended until a non-
- 25 interesting data value is encountered or until the top of the branch is reached, the encoding of another tree is begun. Accordingly, as the encoding method follows the interesting data values of Figure 23 from octave 2 to octave 1 to octave 0, the edge is followed from low to
- 30 high frequency resolution and an increasingly better approximation to the spatial position and shape of the edge is made. Conversely, if at any stage, a non-interesting data value is found, the search is terminated for data values above that non-interesting data value.
- 35 The higher frequency data values of the tree above a non-interesting data value are assumed to be non-interesting because the corresponding low frequency data

values did not indicate the presence of an edge at this location. Any interesting data values that do exist in the higher frequency bands above a non-interesting data value in a low frequency band are rejected as noise.

The one-dimensional tree structure of Figure 23 is 5 encoded as follows. The low frequency components carry visually important information and are therefore always considered to be "interesting". The method of encoding therefore starts with low frequency component A. 10 data value is encoded. Next, the octave 2 data value B is tested to determine if it represents an edge or an edgelike feature which is "interesting" to the human visual system. Because data value B is interesting, a token is generated representing that the bits to follow will 15 represent an encoded data value. Interesting data value B is then encoded. Because this tree has not yet terminated, the method continues upward in frequency. Data value C of octave 1 is then tested. For purpose of this example, data value C is considered to be interesting 20 as are data values A, B, C, D, G, H, J, L and M as illustrated in Figure 23. A token is therefore generated indicating an encoded data value will follow. After the token is sent, data value C is encoded. Because this branch has still not terminated in a non-interesting data 25 value, the method continues upward in frequency. Data value D is tested to determine whether or not it is interesting. Because data value D is interesting, a token is generated and data value D is encoded. Because octave 0 is the highest octave in the decomposition, the encoding 30 method tests the other branch originating from previous interesting data value C. Data value E however tests to be non-interesting. A non-interesting token is therefore generated. Data value E is not encoded and does not appear in the compressed data. With both branches 35 originating at data value C terminated, the method proceeds down in frequency to test the remaining branches originating from the previous interesting data value B.

Data value F is, however, determined to be noninteresting. A non-interesting token is therefore generated and data value F is not encoded and does not appear in the encoded data. Because this branch has 5 terminated, all data values higher in frequency above data value F are considered to be non-interesting. A decoding device receiving the sequence of encoded data values and tokens can determine from the non-interesting token that all corresponding higher frequency data values were 10 considered to be non-interesting by the encoding device. The decoding device can therefore write the appropriate data values as non-interesting and write zeroes to these locations obviating the need for the encoding device to transmit each non-interesting data value above F. With 15 the first tree encoded, the method proceeds to the next low frequency component, data value G. This is a low frequency component and therefore is always considered to be interesting. Data value G is therefore encoded. method then proceeds to the next tree through blocks H, I, 20 J, K and L in that order generating interesting and noninteresting tokens and encoding interesting data values. Similarly, after the second tree is terminated, low frequency component data value M is encoded. Data value N is determined to be non-interesting so a non-interesting 25 token is sent and the encoding of the third tree is terminated.

In accordance with another embodiment of the present invention, a two-dimensional extension of the one-dimensional case is used. Rather than using binary trees, 30 four branch trees are used. However, to create a practical image encoding method there are also real world factors to take into account. Using a single data value to predict whether the remainder of the tree is zero, is unreliable when dealing with noisy image data. A small 35 two-by-two block of data values is therefore used as the node element in the tree structure of the two-dimensional embodiment. A decision as to whether or not an edge is

present is based on four data values which is more reliable than a decision based on single data value.

Figure 25 illustrates a tree structure representing a portion of the decomposition of Figure 18. 5 decomposition of Figure 18 may extend farther to the right and farther in a downward direction for larger twodimensional arrays of image data values. Similarly, the tree structure of Figure 25 may extend farther to the right for larger arrays of data values. Figure 25 10 represents a decomposition only having octave 0 and 1 high frequency components. In the event that the decomposition had additional octaves of high frequency components, the tree structure would extend further upward. In contrast to the binary tree structure of Figure 23, the tree 15 structure of Figure 25 is a four branch tree. The two-bytwo block of four octave 1 data values HHHG is the root of a tree which extends upward in frequency to four HG twoby-two blocks. If another octave of decomposition were performed, another level of octave 2 high frequency two-20 by-two blocks would be inserted into the tree structure. Four HHHG octave 1 two-by-two blocks would, for example, have a single octave 2 HHHHHG block beneath them. frequency component would be denoted HHHHHH.

Figure 26 is a pictorial representation of the

25 decomposition of the tree structure of Figure 25. As
explained above with respect to Figure 15, the actual data
values of the various denoted blocks are distributed
throughout the two-dimensional array of data values. The
two numbers separated by a comma in each of the boxes of

30 Figure 25 denote the row and column of a data value of the
two-dimensional array of Figure 18, respectively. Using
this tree structure, it is possible to search through the
transformed data values of Figure 18 encoding interesting
two-by-two blocks of data values and ignoring non35 interesting two-by-two blocks.

To describe how the two dimensional encoding method uses the tree structure to search through a decomposition,

some useful definitions are introduced. First an image decomp is defined with dimensions WIDTH by HEIGHT decomposed to number OCTS of octaves. A function Access is defined such that given some arguments, the function Access outputs the memory address of the specified data value in the wavelet decomposition decomp:

```
address = Access (oct, sub, x, y);
```

oct is the octave of the data value sought and is an integer value between O (the highest octave) and OCTS-1

10 (the number of octaves of transformation OCTS minus one).

Sub indicates which of the HH, HG, GH or GG bands of the decomposition it is that the data value sought is found.

The use of sub = HH to access the low pass data values is only valid when the value of oct is set to that of the lowest octave. The co-ordinates x and y indicate the spatial location from the top left hand corner of the subband specified by oct and sub. The range of valid values of x and y are dependent on the octave being accessed. x has a range of {O...WIDTH/2^{cx+1}}. y has a range of {O...

Given the function Access and a wavelet decomposition, a two-by-two block of data values can be read by the function ReadBlock.

```
block = ReadBlock (decomp, oct, sub, x, y) {

block[0][0] = decomp[Access(oct, sub, x, y)];

block[0][1] = decomp[Access(oct, sub, x+1, y)];

block[1][0] = decomp[Access(oct, sub, x, y+1)];

block[1][1] = decomp[Access(oct, sub, x+1, y+1)];
}
```

The wavelet decomposition is passed to the function ReadBlock via the variable decomp. The two-by-two block of data values is returned through the variable block. Once a two-by-two block of data values is read, a decision is made as to whether the two-by-two block is visually "interesting" and should therefore be encoded or whether it is not and hence should be discarded. The decision is made by a function called Threshold. The arguments of the function Threshold are block, oct and sub. Threshold returns a boolean value True if the block is "interesting" and False if the block is "non-interesting".

If the block is determined to be interesting by the function threshold, it is encoded using a function called EncodeBlock. A function SendToken inserts a token before the encoded block to inform a decoding device which will later decode the compressed data whether the block to follow the token has been encoded (i.e. BlockNotEmpty) or 15 has not been encoded (i.e. BlockEmpty). If a block is determined to be interesting, then a BlockNotEmpty token is sent, and the block is encoded; next the tree structure above the encoded block is ascended to better determine the location of the edge. The tree encoding procedure 20 SendTree is therefore defined recursively as follows:

```
SendTree (decomp, oct, sub, x, y, Q) {
    block = ReadBlock (decomp, oct, sub, x, y);
    If Threshold (block, oct, sub, Q) {
        SendToken (BlockNotEmpty);
        EncodeBlock (block, oct, sub, Q);
        If (oct >0) {
            SendTree (decomp, oct-1, sub, 2*x, 2*y, Q);
            SendTree (decomp, oct-1, sub, 2*(x+1), 2*y, Q);
            SendTree (decomp, oct-1, sub, 2*x, 2*(y+1), Q);
            SendTree (decomp, oct-1, sub, 2*x, 2*(y+1), Q);
        }
        else SendToken (BlockEmpty);
}
```

The procedure SendTree is only used to encode high-35 pass component data values. In procedure SendTree (decomp, oct, sub, x, y, Q), if the two-by-two block accessed by ReadBlock is determined to pass the threshold test, then SendTree (decomp, oct-1, sub 2*X, 2*y, Q) is used to test one of the next higher two-by-two blocks in the decomposition tree.

The low-pass data values are not considered to form part of the tree structure. The low-pass data values are encoded using another procedure SendLPF. In addition, the low-pass values are encoded using a different technique than that used in EncodeBlock, so a new procedure EncodeBlockLPF is required.

```
SendLPF (decomp, x, y, Q) {
    block = Readblock (decomp, OCTS-1, HH, x, y);
    EncodeBlockLPF (block, OCTS-1, Q);
15 }
```

Accordingly, to encode the entire image, SendLPF is applied to all the block locations within the low pass band and SendTree is applied to the all the block locations in the HG, GH and GG bands, within the lowest octave. A procedure SendDecomp is therefore defined that encodes the entire image decomposition:

```
SendDecomp (decomp, Q) {
    For (y=0; y<HEIGHT/2<sup>octs</sup>; y=y+2)
    For (x=0; x<WIDTH/2<sup>octs</sup>; x=x+2) {

SendLPF (decomp, x, y, Q);
    SendTree (decomp, OCTS-1, HG, x, y, Q);
    SendTree (decomp, OCTS-1, GH, x, y, Q);
    SendTree (decomp, OCTS-1, GG, x, y, Q);
}
```

Accordingly, the above functions define a method for encoding wavelet decomposed images. In terms of speed of encoding for real-world images, many of the trees are

terminated within the initial octaves so much of the decomposition is not examined. Due to this termination of many trees in the initial octaves, many data values need not be encoded which results in reducing the memory bandwidth and block processing required to implement the compression/decompression method. Provided the functions Threshold, EncodeBlockLPF and Access require only simple calculations, the decomposed data values are rapidly encoded.

To implement the function Access, a table containing all the addresses of the data values of the two-dimensional tree decomposition may be accessed using the variables x, y, sub and oct. For a small image having a small number of data values, this table lookup approach is reasonable. For images having, for example, approximately 80 different values of x, 60 different values of y, four different values of sub, and 3 or 4 values for oct, this table would contain approximately 150,000 10-bit locations. A less memory intensive way of determining the 20 same X and Y addresses from the same variables is desirable.

In accordance with one embodiment of the present invention, a function is used to determine the X and Y addresses from the variables x, y, sub and oct. Address 25 X, for example, may be determined as follows:

$$X = ((x << 1) + (sub >> 1)) << oct$$

where << denotes one shift to the right of value x and where >> denotes one shift to the left.

Address Y, for example, may be determined as follows:

$$Y = ((y << 1) + (1 & sub)) << oct$$

where & denotes a bit-wise AND function.

In a high performance system, the function Access may be implemented according to the following method. The

WO 94/23385 PCT/GB94/00677

- 56 -

recursive function call and the table lookup methods described above are often too slow to implement in real time software or in hardware. Figures 27 and 28 illustrate how the tree decomposition of Figure 25 is 5 traversed in order to generate tokens and encode two-bytwo blocks of data values. The X and the Y in Figures 27 and 28 denote coordinate addresses in the two-dimensional matrix of Figure 18. In order to traverse the tree of the decomposition of Figure 25, it is necessary to be able to 10 determine the X and Y addresses of the data values represented in Figure 25. Figure 27 illustrates how the X and Y address of a two-by-two block of data values are determined for those two-by-two blocks of data values located in octave 0 of the decomposition of Figure 25. 15 Similarly, Figure 28 illustrates how the X and Y addresses of the three two-by-two blocks of data values in octave 1 of the decomposition as well as the one two-by-two block of data values of the low pass component of the decomposition of Figure 25 are determined. X as well as Y 20 are each functions of oct, TreeRoot, and sub. The values of sub, and sub, are determined by the sub-band of the twoby-two block of data values sought.

Figure 29 is a chart illustrating the values of sub, and sub, for each sub-band of the decomposition. If, for 25 example, a two-by-two block of data values is sought in the HH band, then the values of sub, and sub, are 0 and 0, respectively. The values TreeRoot, and TreeRoot, together denote the particular tree of a decomposition containing the particular two-by-two block of the data values sought.

In Figures 27 and 28, the rectangles represent digital counters. The arrows interconnecting the rectangles indicate a sequence of incrementing the counters. For example, the right most rectangle in Figure 27, which is called counter C1, has a least significant bit represented in Figure 27 as bit C1, and a most significant bit represented as bit C1, Similarly, the next rectangle to the left in Figure 27 represents a

digital counter C2 having two bits, a least significant bit C2, and a most significant bit C2,. The structure of the X, Y address depends on the octave in which the two-by-two block of data values being sought resides. To generate the X, Y address in octave oct = 1, the counter C1 is not included, the sub, and sub, bits indicating the sub-band bits are shifted one place to the left, and the least significant bits are filled with zeros. The incrementing of the counters in Figure 28 proceeds as illustrated by the arrows.

To determine the X and Y addresses of the four data values of the low pass component HHHH of Figure 25, Figure 28 is used. Because the two-by-two block of data values being sought is a two-by-two block of the low pass 15 component, the values of sub, and sub, are 0, 0 as required by the table of Figure 29. The C2 counter of Figure 28 increments through the four possible values of C2, and C2, to generate the four addresses in the two-by-two block of data values of the HHHH in the low pass component of 20 Figure 25. The value of TreeRoot, and TreeRoot, are zeroes because this is the first tree of the decomposition. subsequent trees of the decomposition, TreeRoot, and TreeRoot, are incremented as illustrated by the arrows in Figure 28 so that the X and Y addresses of the other two-25 by-two blocks of data values in the low pass component of the tree decomposition can be determined. After this HHHH two-by-two block of data values is located, the four data values are encoded and the search through the tree structure proceeds to the two-by-two block of data values 30 in octave 1 denoted HHHG in Figure 25. To determine the X and Y addresses of the four data values of this two-by-two block, the value of bits sub, and sub, are changed in accordance with Figure 29. Because this two-by-two block is in the HG sub-band, the values of sub, and sub, are 0 35 and 1, respectively. The C2 counter is then incremented through its four values to generate the four addresses of the four data values in that block. Supposing, that this

WO 94/23385 PCT/GB94/00677

- 58 -

two-by-two block is determined to be "interesting" then an interesting token is sent, each of the four data values of the block are encoded, and the tree is then ascended to the two-by-two block of data values in octave 0 denoted These four addresses are determined in accordance with Figure 27. Because the sub-band is sub-band HG, the values of the bits sub, and sub, are 0 and 1, respectively. Counter C1 is then incremented so that the four addresses illustrated in the two-by-two block octave 0 HG#1 of 10 Figure 25 are generated. If the two-by-two block is interesting, then the interesting token is sent and the four data values are encoded. If the two-by-two block is determined not to be interesting, then a non-interesting token is sent and the four data values are not encoded. 15 The search through the tree structure of the decomposition then proceeds to octave 0 block HG#2. After the four addresses of the octave 0 block HG#1 are generated, the C2. bit of the C2 counter is incremented in accordance with the arrows shown in Figure 27. Accordingly, the octave 0 20 block HG#2 is addressed when once again the C1 counter increments through its four states. If the data values of this two-by-two block are determined to be "interesting", an interesting token is sent followed by the encoded data values. If the data values of the two-by-two block are 25 determined to be non-interesting, then a non-interesting token is sent. After all the search of the four two-bytwo blocks of the octave 0 HG sub-band are searched, then that HG tree is terminated and the search proceeds to determine the four addresses of the four data values of 30 the octave 1 HHGH two-by-two block. In accordance with this technique, it is possible to traverse the structure of the decomposition and determine the addresses of any two-by-two block in any octave or any sub-band with minimum overhead. Moving between consecutive addresses or 35 descending trees is a simple operation when compared to the snaking address path used by other compression methods

such as JPEG.

When implemented in software, this technique enables real time compression and decompression whereas other techniques may be too slow. If implemented in hardware, this technique provides for a reduced gate count and an 5 efficient implementation. Although this example shows one way of traversing the tree structure of wavelet transform decomposition, it is possible to traverse the tree structure in other ways simply by changing the control structure represented in Figures 27 and 28 to allow for a 10 different traversal of the tree structure. For example, all of the low pass HHHH blocks can be located and encoded first followed by all of the HHGG tree of the decomposition, and then all of the HHGH trees, and then all of the HHGG trees.

15 QUANTIZATION

Each data value of each two-by-two block of the tree decomposition which is determined to be "interesting" is quantized and then Huffman encoded. A linear mid-step quantizer with double-width-0 step is used to quantize 20 each of the data values. Figure 30 is an illustration of the quantization of a 10-bit twos complement data value. The range of the 10-bit data value to be quantized ranges from -512 to 511 as illustrated by the numbers above the horizontal line in Figure 30. This range is broken up 25 into a plurality of steps. Figure 31 represents one such step of data values which extends from 128 to 256 in Figure 30. All incoming data values having values between 128 and 255 inclusive are quantized by dividing the data value by the value qstep. Accordingly, the data value A 30 having a value of 150 as illustrated in Figure 31 is divided by the qstep value 128 and results in a qindex number of 1. Integer division is used to generate gindex and the fractional part of the remainder is discarded. Once the qindex number is determined, the qindex number is 35 Huffman encoded. An overall Q value is sent once per frame of compressed data values. The value qstep is

determined from the overall Q value as described below.

To inverse quantize the qindex number and the qstep value to determine the value of the transformed data values before inverse transformation, the device decoding the incoming quantized values calculates the value of qstep using the value of Q according to a method described below. Once the value of qstep in determined, qindex for a given data value is multiplied by qstep.

In the example of Figure 31, qindex value 1 times

10 qstep 128 results in an inverse quantized value of 128.

If this inverse quantized value of 128 were used, however, all the data values in the step 128 through 255 would be inverse quantized to the value of 128 at the left end of the step. This would result in unacceptably large errors.

15 On the other hand, if all the data values in the range of Figure 31 were inverse quantized to the mid-step value 191, then less error would result. Accordingly, an inverse quantized value qvalve can be calculated from qindex and qstep as follows:

$$20 \ qvalue(qindex,qstep) = \begin{cases} qindex*qstep-\left(\frac{qstep}{2}-1\right) \ if \ qindex<0 \\ 0 \ if \ qindex=0 \\ qindex*qstep+\left(\frac{qstep}{2}-1\right) \ if \ qindex>0 \end{cases}$$

The human visual system, however, has different sensitivities to quantization errors depending upon the particular sub-band containing the quantized data values. The human visual system performs complex non-linear processing. Although the way the human visual system relates image intensities to recognizable structures is not well understood, it is nevertheless important to take advantage of as much information about the human visual system as possible in order to maximize compression ratio versus picture quality. The wavelet transform approximates the initial image processing performed by the human brain. Factors such as spatial frequency response and Weber's Law can therefore be applied directly to the

wavelet transformed data values because the transformed data values are in a convenient representation.

Figure 32 shows the sensitivity of the human eye to spatial frequency. Spatial frequency is measured in 5 cycles c per visual angle θ. A screen is positioned at a distance d from an observer as illustrated in Figure 33. A light of sinusoidally varying luminance is projected onto the screen. The spatial frequency is the number of luminance cycles c per visual degree θ at distance d.

10 Note from Figure 32 that the sensitivity of the human eye varies with spatial frequency. Accordingly, the value of qstep is varied depending on the octave and sub-band of the data valve being quantized. The qstep at which a data valve is quantized is determined from the variables 15 oct, sub and Q for that data valve as follows:

 $qstep(oct, sub, Q) = Q * hvs_factor(oct, sub)$

$$hvs_factor(oct, sub) = \begin{cases} \sqrt{2} & \text{if } sub=GG \\ 1 & \text{otherwise} \end{cases} = \begin{cases} 1.00 & \text{if } oct=0 \\ 0.32 & \text{if } oct=1 \\ 0.16 & \text{if } oct=2 \\ 0.10 & \text{if } oct=3 \end{cases}$$

The scaling factors 1.00, 0.32, 0.16 and 0.10 relate to the spatial frequency scale of Figure 32 to take into 20 account the frequency dependent sensitivity of the human eye.

It is to be understood that scaling factors other than 1.00, 0.32, 0.16 and 0.10 could be used. For example, other scaling factors can be used where the 25 quantizer is used to compress audio data which is received by the human ear rather than by the human eye. Moreover, note that the sub-band GG is quantized more heavily than the other sub-bands because the sub-band GG contains diagonal information which is less important to the human 30 eye than horizontal and vertical information. This method can also be extended down to the level of two-by-two blocks of data values to further tailor the degree of quantization to the human visual system. The function

hvs_factor which has only two parameters in the presently described embodiment is only one embodiment of the present invention. The function hvs_factor, for example, can take into account other characteristics of the human visual system other than oct and sub, such as the luminance of the background and texture masking.

THRESHOLDING

For each new two-by-two block of data values in the tree decomposition, a decision must be made as to whether 10 the block is "interesting" or "non-interesting". This can be done by the function threshold:

threshold(block, limit) = limit >
$$\sum_{y=0}^{1} \sum_{x=0}^{1} |block[y][x]|$$
 (equ. 60)

The sum of the absolute values of the data values of the block block is determined as is represented by the double summation to the right of the less than sign and this value is compared to a threshold value limit.

"Interesting" blocks are those blocks, for which the sum of the absolute values of the four data values exceeds the value limit, whereas "non-interesting" blocks are those blocks for which the sum is less than or equal to the value limit.

The value limit takes into account the variable quantizer step size qstep which varies with octave. For 25 example, a two-by-two block of data values could be determined to pass the test threshold, but after quantizing by qstep could result in four zero quantized values. For example, all data values between -128 and 127 are quantized to have a quantized qindex of zero as is 30 shown in Figure 30 even if some of those data values are determined to correspond with an "interesting" two-by-two block. For this reason, the value limit is calculated according to the equation:

limit = 4*Bthreshold*qstep (equ. 61)

In this equation "Bthreshold" is base threshold image factor. In the presently described example, this base threshold is equal to 1.0. The value of 1.0 for the base 5 threshold Bthreshold was determined through extensive experimentation on test images. The factor 4 in equation 61 is included to account for the fact that there are four data values in the block under consideration. In this way blocks are not determined to be interesting, the data 10 values for which the quantizer will later reduce to zeros. This weighted threshold factor limit also reduces the number of operations performed in the quantizer because a fewer number of data values are quantized.

HUFFMAN CODING

The wavelet transform produces transformed data values whose statistics are vastly different from the data values of the original image. The transformed data values of the high-pass sub-bands have a probability distribution that is similar to an exponential or Laplacian 20 characteristic with mean zero.

Figure 34 shows the distribution of high pass data values in a four octave wavelet decomposition of the test image Lenna. Figure 35 shows the distribution of the data values of the test image Lenna before wavelet transforma-

- 25 tion. The low-pass component data values have a flat distribution that approximates the distribution of luminance and chrominance values in the original image. The high and low pass data values are encoded differently for this reason.
- The low pass component data values are encoded by the function *EncodeBlockLPF* as follows:

```
EncodeBlockLPF ( block, OCT-1, Q) {
    Output ( block[0][0]/qstep( OCT-1, HH, Q));
    Output ( block[0][1]/qstep( OCT-1, HH, Q));

Output ( block[1][0]/qstep( OCT-1, HH, Q));
```

WO 94/23385 PCT/GB94/00677

- 64 -

```
Output ( block[1][1]/qstep( OCT-1, HH, Q));}
```

After encoding, the low-pass data values are quantized and output into the compressed data stream. The low pass data values are not Huffman encoded.

The high frequency component data values which pass the threshold test are quantized and Huffman encoded to take advantage of their Laplacian distribution. Function EncodeBlock performs the quantization and the Huffman encoding for each of the four data values of an 10 interesting high frequency component block block. In the function EncodeBlock, the variable sub is provided so that when function qstep is called, different quantization qstep values can be used for different high frequency component sub-bands. The function huffman performs a 15 table lookup to a fixed Huffman code table such as the table of Table 3. The function EncodeBlock is defined as

```
EncodeBlock (block, oct, sub, Q) {
    Output(huffman(block[0][0]/qstep(oct, sub, Q)));

Output(huffman(block[0][1]/qstep(oct, sub, Q)));

Output(huffman(block[1][0]/qstep(oct, sub, Q)));

Output(huffman(block[1][1]/qstep(oct, sub, Q)));
}
```

follows:

	qindex	Huffman code
	-38512	1 1 0 0 0 0 0 0 1 1 1 1 1 1 1 1
	-2237	1 1 0 0 0 0 0 0 1 1 1 1 (qindex -22)
	721	1 1 0 0 0 0 0 0 (qindex -7)
5	6	1 1 0 0 0 0 0 1
	•	•
	•	
	-2	1 1 0 1
10	-1	1 1 1
	0	0
	1	1 0 1
	2	1 0 0 1
	•	
15	•	
	6	1000001
	7 21	1 0 0 0 0 0 0 0 (qindex -7)
	22 37	1 0 0 0 0 0 0 0 1 1 1 1 (qindex -22)
20	38 511	100000001111111

Table 3

The second bit from the left in the Huffman code of Table 3 is a sign bit. The value |qindex|-7 is represented with 4 bits in the case 7 ≤ |qindex|≤ 21. The 25 value |qindex|-22 is represented with 4 bits in the case 22 ≤|qindex|≤ 37).

ENCODING OF TOKENS

At high compression ratios the number of bits in the compressed data stream used by tokens may be reduced by 30 amalgamating groups of "non-interesting" tokens. This can be achieved by introducing new tokens. In accordance with one embodiment of the present invention, two new tokens, OctEmpty and OctNotEmpty are used. For a high pass component block in a tree above octave zero, there are 35 four branches. The additional pair of tokens indicate

WO 94/23385

whether all four are non-interesting. If all four are non-interesting, only a single OctEmpty token need be sent. Otherwise, an OctNotEmpty token is generated before the four branches are encoded. The particular token 5 scheme described above was selected more to simplify the hardware and software implementations than it was to achieve in the best compression ratio possible. Other methods of representing relatively long sequences of token bits in the compressed data stream using other tokens 10 having a relatively fewer number of bits may be used in place of the tokens OctEmpty and OctNotEmpty to achieve higher compression ratios.

VIDEO ENCODING AND DECODING

In comparison with the coding of a still image, the

15 successive images of a video sequence typically contain
much redundant information. The redundancy of this
information is used to reduce the bit rate. If a location
in a new frame of the video contains the same or
substantially the same information as a corresponding

20 location in the previous old frame of video, that portion
of the new frame need not be encoded and introduced into
the compressed data. This results in a reduction in the
total number of bits in the encoded bit stream.

Figure 36 illustrates a video encoder 31 and a video
25 decoder 32. A video input signal is transformed by a
forward wavelet transform block 33, the output of which is
written to a new frame store 34. The first frame of video
information in the new frame store 34 is referred to as
the new frame because no previous frame exists in the old
30 frame store 35 for containing an old frame. A comparison
tree encoder 36 therefore generates tokens and transformed
data values as described above from the data values output
from new frame store 34. The transformed data values are
quantized by quantizer 37 into qindex levels. These
35 qindex levels are then Huffman coded by the Huffman
encoder 38. The resulting encoded data values are then

WO 94/23385 PCT/GB94/00677

combined with the tokens in buffer 38A to form a decompressed data bit stream 39.

An essential part of this method is that the old frame present in the video encoder 31 is exactly the same 5 as the old frame 40 present in the video decoder 32. This allows the decoder 32 to be able to correctly decode the encoded bit stream 39 due to the fact that the encoded bit stream contains differences between new and old images and due to the fact that parts of the new frame are not sent 10 due to compression. An inverse quantizer 41 is therefore provided in the video encoder 31 to inverse quantize the qindex levels and to store the old frame as sent into old frame store 35 for future comparison with the next frame of the video input signal.

- In the video decoder 32, the compressed data stream 39 is received by a buffer 42. The tokens are separated from the Huffman encoded qindex levels. The Huffman encoded qindex levels are supplied to a Huffman decoder 43, the output of which is supplied to an inverse
- 20 quantizer 44. The output of the inverse quantizer 44 is written into old frame store 40 under the control of the comparison tree decoder 45. Comparison tree decoder 45 determines what is written into the old frame store 40, depending in part on the tokens received from buffer 42.
- 25 Once a new frame of transformed data values is present in old frame store 40, an inverse wavelet transform 46 inverse transforms that frame of transformed data values into a corresponding video output signal. To prevent the inverse wavelet transform 46 from overwriting and
- 30 therefore corrupting the contents of old frame store 40 when it reconstructs data values corresponding to the original new frame data values, an intermediate frame store 47 is maintained.

The octave one HHHG, HHGH, HHGG, and HHHH from Figure 35 25 are read from the old frame store 40 by the inverse wavelet transform 46 to perform the octave 1 inverse transform as described above. However, the resulting

octave 0 HH sub-band, output from the inverse wavelet tranform 46 is now written to the intermediate frame store 47, so as not to corrupt the old frame store 40. For the octave 0 inverse wavelet transform, the HG, GH, and GG sub-bands are read from the old frame store 40, and the HH sub-band is read from the intermediate frame store 47, to complete the inverse wavelet transform.

When the second frame of compressed video data 39 is received by the video decoder 32, the tokens received by 10 the comparison tree decoder 45 are related to the contents of the previous frame of video information contained in old frame store 40. Accordingly, the video decoder 32 can reconstruct the latest frame of video data using the contents of the frame store 40 and the data values encoded 15 in the compressed data stream 39. This is possible because the compressed data stream contains all the information necessary for the video decoder 32 to follow the same traversal of the tree of the decomposition that the encoder used to traverse the tree in the generation of 20 the compressed data stream. The video decoder 32 therefore works in lock step with the video encoder 31. Both the encoder 31 and the decoder 32 maintain the same mode at a corresponding location in the tree. When the encoder 31 determines a new mode, it incorporates into the 25 compressed data stream 39 a corresponding token, which the video decoder 32 uses to assume that new mode.

Figure 37 illustrates the modes of operation of one possible embodiment of the present invention. To explain the operation of the video encoder 31 and the video decoder 32, an example is provided. The initial frame of the video sequence is processed by the video encoder 31 in still mode. Still mode has three sub-modes: STILL, VOID_STILL, and LPF_STILL. The low pass two-by-two blocks of data values of the decomposition cause the comparison tree encoder 36 of video encoder 31 to enter the LPF_STILL sub-mode. In this sub-mode, the four data values of the two-by-two block are quantized but are not Huffman

WO 94/23385 PCT/GB94/00677

- 69 -

encoded. Similarly, no token is generated. The successive low pass component two-by-two blocks of data values are successively quantized and output into the compressed data stream 39.

Next, the lowest frequency octave of one of the subbands is processed by the comparison tree encoder 36.

This two-by-two block of data values corresponds with
block HHHG illustrated in Figure 25. The four data values
of this two-by-two block are tested against the threshold

limit to determine if it is "interesting". If the
two-by-two block HHHG is interesting, then a single bit
token 1 is generated, as illustrated in Figure 37, the
mode of the comparison tree encoder remains in STILL mode,
and the four data values of the two-by-two block HHHG are
successively quantized and encoded and output into the
compressed data stream 39.

For the purposes of this example, block HHHG is assumed to be interesting. The tree structure of Figure 25 is therefore ascended to octave 0 two-by-two block 20 HG#1. Because the comparison tree encoder 31 remains in the STILL mode, this block is encoded in the STILL mode. The four data values of block HG#1 are tested to determine whether or not they are interesting. This sequence of testing the successive blocks of the tree structure is repeated as described above.

After the traversal of the four octave 0 sub-blocks HG#1, HG#2, HG#3 and HG#4, the comparison tree encoder 36 proceeds in the tree structure to the two-by-two block of data values in octave 1, block HHGH. For purposes of this example, this two-by-two is non-interesting. After the comparison tree encoder 36 reads the four data values, the result of the threshold test indicates a non-interesting two-by-two block. As illustrated in Figure 37, the encoder 31 which is in the still mode now generates a single bit token 0 and the comparison tree encoder 36 enters the VOID_STILL sub-mode. Although no additional information is output into the compressed data stream 39,

WO 94/23385 PCT/GB94/00677

- 70 -

the comparison tree encoder 36 proceeds to write 0's into the four locations of the two-by-two block HHGH, as well as all the locations of the two-by-two blocks in the tree above the non-interesting two-by-two block HHGH. 5 example of Figure 25, the comparison tree encoder 36 writes 0's into all the addresses of blocks HHGH, GH#1, GH#2, GH#3 and GH#4. This zeroing is performed because the video decoder 32 will not be receiving the data values corresponding to that tree. Rather, the video decoder 32 10 will be receiving only a non-interesting token, a single bit 0. The video decoder 32 will therefore write zeros into frame store 40 in the remainder of the corresponding In order to make sure that both the video encoder tree. 31 and the video decoder 32 have exactly the same old 15 frame 35 and 40, the video encoder too must zero out those non-interesting blocks.

After the first frame of video data has been encoded and sent in STILL mode, the next frame of video data is processed by the video encoder 31. By default, the 20 encoder now enters SEND mode. For lowpass frequency component two-by-two blocks, the video encoder 31 enters the LPF_SEND mode as illustrated in Figure 37. encoding of such a lowpass component two-by-two block corresponds with the encoding of two-by-two block HHHH in 25 Figure 25. However, now the comparison tree encoder 36 has both a new frame in frame store 34 as well as an old frame in frame store 35. Accordingly, the comparison tree encoder 36 determines the arithmetic difference of the respective four data values in the new frame from the four 30 data values in the old frame at the corresponding position and compares the sum of those differences with a compare threshold. The compare threshold, compare, is calculated from a base compare threshold "Bcompare" as in the case of the previous threshold which determines which blocks are 35 interesting, similar to equations 60 and 61. If the sum of the differences is less than the compare threshold, then the video encoder 31 sends a single bit token 0 and

remains in the LPF_SEND mode, as illustrated in Figure 37. The video encoder 31 does not transmit any data values corresponding to the lowpass frequency component two-by-two block.

If, on the other hand, the sum of the arithmetic differences exceeds the compare threshold, then a single bit token 1 is generated, as illustrated in Figure 37. In this case, the video encoder 31 sends the arithmetic differences of each of the successive four data values of the new frame versus the old frame to the quantizer 37 and then to the Huffman encoder 38. The arithmetic differences are encoded and sent rather than sending the actual data values because this results in fewer bits due to the fact that the two blocks in the new and old frames are quite similar under normal circumstances.

When the video encoder 31 proceeds to encode the octave 1 sub-band HHHG, as illustrated in Figure 25, the video encoder 31 enters the SEND mode, as illustrated in Figure 37. In this mode, the comparison tree encoder 36 compares the data values of the new two-by-two block with the data values of the old two-by-two block and performs a series of arithmetic operations to generate a series of flags, as illustrated in Figure 38. Based on these flags, the video encoder 31 generates a 2-bit token and enters one of four new modes for that two-by-two block. If, for example, the two-by-two block HHHG in Figure 25 is received by the video encoder 31, then flags ozflag, nzflag, new_z, noflag, motion, origin, and no_z are determined. The values of these flags are determined as:

$$nz = \sum_{x=0}^{1} \sum_{y=0}^{1} |new[x][y]|$$
 (equ. 62)

$$no = \sum_{x=0}^{1} \sum_{y=0}^{1} |new[x][y] - old[x][y]|$$
 (equ. 63)

oz =
$$\sum_{x=0}^{1} \sum_{y=0}^{1} |old[x][y]|$$
 (equ. 64)

Based on the values of these flags, the new mode for 10 the two-by-two block HHHG is determined, from Figure 38.

If the new mode is determined to be the SEND mode, the 2-bit token 11 is sent as indicated in Figure 37. The arithmetic differences of the corresponding four data values are determined, quantized, Huffman encoded, and 15 sent into the compressed data stream 39.

In the case that the flags indicate the new mode is STILL_SEND, then the 2-bit token 01 is sent and the new four data values of the two-by-two block are quantized, Huffman encoded, and sent. Once having entered the 20 STILL_SEND mode, the video encoder 31 remains in the STILL_SEND mode until the end of the tree has been reached. In this STILL_SEND mode, a single bit token of either 1 or 0 precedes the encoding of each block of data values. When the VOID mode is entered from STILL_SEND 25 mode, the video encoder 31 generates a single bit 0 token, then places zeros in the corresponding addresses for that two-by-two block, and then proceeds to place zeros in the addresses of data values of the two-by-two blocks in the tree above.

In the event that the flags indicate that the video encoder 31 enters the VOID mode from SEND mode, a 2-bit token 10 is generated and the four data values of that two-by-two block are replaced with zeros. The VOID mode also results in the video encoder 31 placing zeros in all 35 addresses of all data values of two-by-two blocks in the tree above.

In the case that the flags indicate that there is no

WO 94/23385 PCT/GB94/00677

- 73 -

additional information in the tree being presently encoded, namely, the new and the old trees are substantially the same, then a 2-bit token of 00 is generated and the video encoder 31 proceeds to the next 5 tree in the decomposition.

In general, when the video encoder 31 enters VOID mode, the video encoder will remain in VOID mode until it determines that the old block already contains four zero data values. In this case, there is no reason to continue in VOID mode writing zeros into that two-by-two block or the remainder of the blocks in the tree above because it is guaranteed that the old tree already contains zeros in these blocks. This is true because the old tree in frame store 35 has previously been encoded through the inverse quantizer 41.

Because the video decoder 32 is aware of the tree structure of the decomposition, and because the video encoder 31 communicates with the video decoder 32 using tokens, the video decoder 32 is directed through the tree 20 structure in the same manner that the video encoder 31 traverses the tree structure in generating the compressed data stream 39. In this way the video decoder 32 writes the appropriate data values from the decompressed data stream 39 into the corresponding positions of the old data 25 frame 40. The only flag needed by the video decoder 32 is the ozflag, which the video decoder obtains by reading the contents of old frame store 40.

RATE CONTROL

All transmission media and storage media have a
30 maximum bandwidth at which they can accept data. This
bandwidth can be denoted in terms of bits per second. A
standard rate ISDN channel digital telephone line has, for
example, a bandwidth of 64 kbits/sec. When compressing a
sequence of images in a video sequence, depending upon the
35 amount of compression used to compress the images, there
may be a relatively high number of bits per second

generated. This number of bits per second may in some instances exceed the maximum bandwidth of the transmission media or storage device. It is therefore necessary to reduce the bits per second generated to insure that the maximum bandwidth of the transmission media or storage device is not exceeded.

One way of regulating the number of bits per second introduced into the transmission media or storage device involves the use of a buffer. Frames having a high number of bits are stored in the frame buffer, along with frames having a low number of bits, whereas the number of bits per second passing out of the buffer and into the transmission media or storage device is maintained at a relatively constant number. If the buffer is sufficiently large, then it is possible to always achieve the desired bit rate as long as the overall average of bits per second being input into the buffer over time is the same or less than the maximum bit rate being output from the buffer to the transmission media or storage device.

There is, however, a problem associated with large buffers in video telephony. For a large buffer, there is a significant time delay between the time a frame of video data is input into the buffer and time when this frame is output from the video buffer and into the transmission 25 media or storage device. In the case of video telephony, large buffers may result in large time delays between the time when one user begins to speak and the time when another user begins to hear that speech. This time delay, called latency, is undesirable. For this reason, buffer 30 size is specified in the standard H.261 for video telephony.

In accordance with one embodiment of the present invention, a rate control mechanism is provided which varies the number of bits generated per frame, on a frame 35 by frame basis. Due to the tree encoding structure described above, the number of bits output for a given frame is dependent upon the number of trees ascended in

the tree encoding process. The decisions of whether or not to ascend a tree are made in the lowest high frequency octaves of the tree structure. As can be seen from Figure 25, there are relatively few number of blocks in the 5 lowest frequency of the sub-bands, as compared to the number of blocks higher up in the sub-band trees. Given a particular two-by-two block in the tree structure, it is possible to decrease the value of Q in the equation for the threshold limit until that particular block is 10 determined to be "interesting". Accordingly, a particular Q is determined at which that particular block becomes interesting. This process can be done for each block in the lowest frequency HG, GH and GG sub-bands. In this way, a histogram is generated indicating a number of 15 two-by-two blocks in the lowest frequency of the three sub-bands which become interesting at each particular value of Q.

From this histogram, a relationship is developed of the total number of two-by-two blocks in the lowest 20 frequency of the three sub-bands which are interesting for a given value of Q. Assuming that the number of blocks in the lowest frequency octave of the three sub-bands which are interesting for a given value of Q is representative of the number of bits which will be generated when the 25 tree is ascended using that given value of Q, it is possible to determine the value of Q at which a desired number of bits will be generated when that frame is coded with that value of Q. Furthermore, the greater the threshold is exceeded, the more bits may be needed to 30 encode that tree. It is therefore possible to weight by Q the number of blocks which are interesting for a given value of Q. Finally, the Q values so derived should be averaged between frames to smooth out fluctuations.

The encoder model RM8 of the CCITT Recommendation
35 H.261 is based on the DCT and has the following
disadvantages. The rate control method used by RM8 is a
linear feedback technique. Buffer fullness is

proportional to Q. The value of Q must be adjusted after every group of blocks (GOB) to avoid overflow or underflow effects. This means that parts of the image are transmitted at a different level quality from other parts.

5 During parts of the image where little change occurs, Q drops which can result in uninteresting areas being coded very accurately. The objects of interest are, however, usually the moving ones. Conversely, during the coding of areas of high activity, Q rises creating large errors in moving areas. When this is combined with a block based transform, the errors can become visually annoying.

The method of rate control described in connection with one embodiment of the present invention uses one value of Q for the whole frame. The value of Q is only adjusted between frames. All parts of an image are therefore encoded with the same value of Q. Moreover, because the tree structure allows a relatively few number of blocks to be tested to determine an estimate of the number of bits generated for a given frame, more intelligent methods of varying Q to achieve an overall desired bit rate are possible than are possible with conventional compression/decompression techniques.

TREE BASED MOTION ESTIMATION

Figure 39 represents a black box 1 on a white

25 background 2. Figure 40 represents the same black box 1 on the same white background 2 moved to the right so that it occupies a different location. If these two frames of Figures 39 and 40 are encoded according to the above described method, there will be a tree in the wavelet

30 decomposition which corresponds with the white-to-black edge denoted 3 in Figure 39. Similarly, there will be another tree in the wavelet decomposition of the image of Figure 40 which represents the white-to-black edge 3' the wavelet decomposition of the image of Figure 40. All of

35 the data values corresponding to these two trees will be determined to be "interesting" because edges result in

interesting data values in all octaves of the decomposition. Moreover, due to the movement of the corresponding edge of black box 1, all the data values of the edges of both of these two trees will be encoded as interesting data values in the resulting compressed data stream. The method described above therefore does not take into account that it is the same data values representing the same white-to-black edge which is present in both images but which is just located at a different location.

Figure 41 is a one dimensional representation of an edge. The corresponding low path component data values are not illustrated in Figure 41. Data values 4, 5, 6, 7, 8, and 9 represent the "interesting" data values of Figure 41 whereas the other data values have low data values which makes those blocks "non-interesting". In the representation of Figure 41, data values 4 and 5 are considered a single two data value block. Similarly, blocks 6 and 7 are considered a single block and blocks 8 20 and 9 are considered a single block. Figure 41, although it is a one dimensional representation for ease of illustration, represents the edge 3 of the frame of Figure 39.

Figure 42 represents the edge 3' shown in Figure 40.

25 Figure 42 indicates that the edge of black box 1 has moved in location due to the fact that the values 19 and 21 which in Figure 41 were in the two data value block 8 and 9 are located in Figure 42 in the two data value block 10 and 11. In the encoding of Figure 42, rather than

30 encoding and sending into the compressed data stream the values 19 and 21, a control code is generated which indicates the new locations of the two values. Although numerous control codes are possible, only one embodiment is described here.

When the two data value block 10 and 11 is tested to determine whether it is interesting or not, the block tests to be interesting. The neighboring blocks in the

old frame are, however, also tested to determine whether the same values are present. In this case, the values 19 and 21 are determined to have moved one two data value block to the right. An "interesting with motion" token is 5 therefore generated rather than a simple "interesting" token. A single bit 1 is then sent indicating that the edge represented by values 19 and 21 has moved to the right. Had the edge moved to the left, a control code of 0 would have been sent indicating that the edge 10 represented by values 19 and 21 moved one location to the left. Accordingly, in the encoding of Figure 42, an "interesting with motion" token is generated followed by a single control code 1. The interesting values 19 and 21 therefore need not be included in the compressed data 15 stream. The video decoder receiving this "interesting with motion" token and this control code 1 can simply copy the interesting values 19 and 21 from the old frame into the indicated new location for these values in the new frame obviating the need for the video encoder to encode 20 and transmit the actual interesting data values themselves. The same token and control codes can be sent for the two data values corresponding to a block in any

Figure 43 represents the motion of the edge 3 of
25 Figure 39 to a new location which is farther removed than
is the new location of black box 1 shown in Figure 40.
Accordingly, it is seen that the values 20 and 21 are
located to the right at the two data value block 12 and
13. In the encoding of this two data value block 12 and
30 13 a token indicating "interesting with motion" is
generated. Following that token, a control code 1 is
generated indicating motion to the right. The video
encoder therefore need not encode the data values 20 and
21 but merely needs to generate the interesting with
35 motion token followed by the motion to the right control
code. When the video encoder proceeds to the two data
values block 14 and 15, the video encoder need not send

one of the octaves 0, 1 or 2.

the "interesting with motion" token but rather only sends
the left control code 0. Similarly, when the video
encoder proceeds to encode the two data value block 16 and
17, the video encoder only sends the left control code 0.

5 The control codes for octaves 0 and 1 do not denote motion
per se but rather denote left or right location above a
lower frequency interesting block of the moving edge.
This results in the video encoder not having to encode any
of the actual data values representing the moved edge in
10 the decomposition of Figure 43.

The one dimensional illustration of Figures 41, 42 and 43 is presented for ease of illustration and explanation. It is to be understood, however, that this method of indicating edge motion is used in conjunction 15 with the above described two dimensional wavelet decomposition such as the two dimensional wavelet decomposition illustrated in Figure 25. The video encoder searches for movement of the data values representing an edge only by searching the nearest neighboring blocks of 20 data values in the old frame. This method can be used to search many neighbors or a few neighbors depending on the application. The counter scheme described in connection with Figures 27 and 28 can be used to determine the locations of those neighboring blocks. Although the edge 25 motion illustrated in connection with Figures 41, 42, and 43 shows the very same data values being moved in the tree structure of the decomposition, it is to be understood that in practice the values of the data values representing the same edge may change slightly with the 30 movement of the edge. The video encoder takes this into account by judging corresponding data values using a motion data value threshold to determine if corresponding data values in fact do represent the same edge. By indicating edge motion and not sending the edge data 35 values themselves it is possible to both increase the compression and also improve the quality of the decompressed image.

SIX COEFFICIENT QUASI-DAUBECHIES FILTERS
The Daubechies six coefficient filters are defined by the six low pass filter coefficients, listed in the table below to 8 decimal places. The coefficients are also defined in terms of four constants, α , β , γ and ϵ , where α = 0.10588942, β = -0.54609641, γ = 2.4254972 and ϵ = 3.0059769.

10		Daubechies coefficients	Alternative representation	Normalized coefficients	Converted Coefficients
	a	0.33267055	1/€	0.2352336	30 128
	þ	0.80689151	γ/ε	0.57055846	$\frac{73}{128}$
	С	0.45987750	-β(α+γ)/ε	0.3251825	128
	-d	-0.13501102	$\beta(1 - \alpha \gamma)/\epsilon$	-0.095467208	<u>-12</u> 128
	-е	-0.08544127	-αγ/ε	-0.060416101	$\frac{-7}{128}$
	£	0.03522629	α/€	0.024908749	128

Table 4

- 15 The coefficients (a, b, c, -d, -e, f) sum to √7. The normalized coefficients sum to 1, which gives the filter the property of unity gain, which in terms of the alternative representation is equivalent to a change in the value of € to 4.2510934. These values can be
- 20 approximated to any given precision by a set of fractions. In the example shown above, each of the normalized values has been multiplied by 128 and rounded appropriately, thus the coefficient a has been converted to $\frac{30}{128}$. Filtering is therefore possible using integer multiplications rather
- 25 than floating point arithmetic. This greatly reduces implementation cost in terms of digital hardware gate count and computer software speed. The following equations show a single step in the filtering process, the outputs H and G being the low and high pass outputs,
- 30 respectively:

$$H_1 = aD_0 + bD_1 + cD_2 - dD_3 - eD_4 + fD_5$$
 (equ. 72)

$$G_1 = -fD_0 - eD_1 + dD_2 + cD_3 - bD_4 + aD_5$$
 (equ. 73)

 H_1 and G_1 are calculated as follows. Each data value Dis multiplied by the relevant integer numerator (30, 73, 41, 12, 7, 3) and summed as shown. The values of H and G 5 are found by dividing the summations by the constant 128. Because 128 is an integer power of 2, the division operation requires little digital hardware to implement and only simple arithmetic shift operations to implement in software. The filters H and G are quasi-perfect

10 reconstruction filters:

$$a+b+c-d-e+f=1$$
 (equ. 74)

$$-f-e+d+c-b+a=0$$
 (equ. 75)

$$a+c-e=\frac{1}{2}$$
 (equ. 76)

$$f-d+b=\frac{1}{2}$$
 (equ. 77)

- 15 Equation 74 guarantees unity gain. Equation 75 guarantees that the high pass filter will generate zero for a constant input signal. Equations 76 and 77 guarantee that an original signal once transferred can be reconstructed exactly.
- The following equations show a single step in the 20 inverse transformation:

$$D_2=2(-eH_0-bG_0+cH_1+dG_1+aH_2-fG_2)$$
 (equ. 78)

$$D_3=2 (fH_0+aG_0-dH_1+cG_1+bH_2-eG_2)$$
 (equ. 79)

As for the forward filtering process, the interleaved 25 H and G data stream is multiplied by the relevant integer numerator and summed as shown. The output D data values are found by dividing the summations by the constant 64, which is also an integer power of 2.

To calculate the first and last H and G values, the 30 filter equations must be altered such that values outside the boundaries of the data stream are not required. For example, if H₀ is to be calcualted using the six coefficient filter, the values D_1 and D_2 would be required. Because

these values are not defined, a different filter is used at the beginning and end of the data stream. The new filters are determined such that the reconstruction process for the first and last two data values is possible. The following 5 pair of equations show the filter used to calculate the first H and G values:

$$H_0 = cD_0 - dD_1 - eD_2 + fD_3$$
 (equ. 80)

$$G_0=dD_0+cD_1-bD_2+aD_3 \qquad (equ. 81)$$

The last H and G values are calculated with:

$$H_5 = aD_8 + bD_9 + cD_A - dD_B$$
 (equ. 82)

$$G_5 = fD_5 - eD_9 + dD_A + cD_B \qquad (equ. 83)$$

In this case, these equations are equivalent to using the non-boundary equations with data values outside the data stream being equal to zero. The following inverse transform boundary filters are used to reconstruct the first two and last two data values:

$$D_0 = 2((c - \frac{b}{\beta})H_0 + (d + \frac{e}{\beta})G_0 + aH_1 - fG_1) \qquad (equ. 84)$$

$$D_1 = 2 \left(\left(\frac{a}{\beta} - d \right) H_0 + \left(c - \frac{f}{\beta} \right) G_0 + bH_1 - eG_1 \right)$$
 (equ. 85)

$$D_{A}=2\left(-eH_{4}-bG_{4}+\left(c-\frac{f}{\beta}\right)H_{5}+\left(d-\frac{a}{\beta}\right)G_{5}\right)$$
 (equ. 86)

$$D_{B}=2\left(fH_{4}+aG_{4}-\left(d+\frac{e}{\beta}\right)H_{5}+\left(c-\frac{b}{\beta}\right)G_{5}\right) \qquad (equ. 87)$$

INCREASING SOFTWARE DECOMPRESSION SPEED

A system is desired for compressing and decompressing video using dedicated digital hardware to compress and 20 using software to decompress. For example, in a video mail application one user uses a hardware compression expansion card for an IBM PC personal computer coupled to a video camera to record a video message in the form of a video message file. This compressed video message file is then 25 transmitted via electronic mail over a network such as a hardwired network of an office building. A recipient user receives the compressed video message file as he/she would receive a normal mail file and then uses the software to

WO 94/23385 PCT/GB94/00677

decompress the compressed video message file to retrieve the video mail. The video mail may be displayed on the monitor of the recipient's personal computer. It is desirable to be able to decompress in software because decompressing in software frees multiple recipients from purchasing relatively expensive hardware. Software for performing the decompression may, for example, be distributed free of charge to reduce the cost of the composite system.

In one prior art system, the Intel Indeo video compression system, a hardware compression expansion card compresses video and a software package is usable to decompress the compressed video. This system, however, only achieves a small compression ratio. Accordingly, video picture quality will not be able to be improved as standard personal computers increase in computing power and/or video bandwidth.

The specification above discloses a method and apparatus for compressing and decompressing video. The 20 software decompression implementation written in the programming language C disclosed in Appendix A only decompresses at a few frames per second on a standard personal computer at the present date. A method capable of implementation in software which realizes faster 25 decompression is therefore desirable.

A method for decompressing video described above is therefore modified to increase software execution speed. Although the b=19/32, a=11/32, c=5/32 and d=3/32 coefficients used to realize the high and low pass forward 30 transform perfect reconstruction digital filters are used by dedicated hardware to compress in accordance with an above described method, the coefficients b=5/8, a=3/8, c=1/8 and d=1/8 are used to decompress in software on a digital computer. The coefficients are determined as shown 35 in the table below.

5

$$a = \frac{1+\sqrt{3}}{8} = .3415(8) = 2.732 \approx \frac{3}{8}$$

$$b = \frac{3+\sqrt{3}}{8} \approx .5915(8) = 4.732 \approx \frac{5}{8}$$

$$c = \frac{3-\sqrt{3}}{8} \approx .1585(8) = 1.268 \approx \frac{1}{8}$$

$$d = \frac{-1+\sqrt{3}}{8} \approx .0915(8) = 0.732 \approx \frac{1}{8}$$

Table 5

An even start inverse transform digital filter in accordance with the present embodiment is:

$$D_0 = 4[(b-a)H_0 + (c-d)G_0]$$
 (equ. 88)

where, for example, D_0 is a first inverse transformed data 10 value indicative of a corresponding first data value of a row of the original image, and where H_0 and G_0 are first low and high pass component transformed data values of a row of a sub-band decomposition.

An odd end inverse transform digital filter in 15 accordance with the present embodiment is:

$$D_B = 4[(c+d)H_5 - (a+b)G_5]$$
 (equ. 89)

where, for example, D_B is a last inverse transformed data value indicative of a corresponding last data value of a row of the original image, and where H₅ and G₅ are last low 20 and high pass component transformed data values of a row of a sub-band decomposition.

An odd interleaved inverse transform digital filter in accordance with the present embodiment is:

$$\frac{D(2x-1)}{2} = \frac{1}{8}H(x-1) - \frac{5}{8}G(x-1) + \frac{3}{8}H(x) + \frac{1}{8}G(x)$$
 (equ. 90)

An even interleaved inverse transform digital filter in accordance with the present embodiment is:

$$\frac{D(2x)}{2} = -\frac{1}{8}H(x-1) + \frac{3}{8}G(x-1) + \frac{5}{8}H(x) + \frac{1}{8}G(x)$$
 (equ. 91)

As indicated by equations 90 and 91, the odd and even interleaved inverse transform digital filters operable on

the same H and G values of the sub-band decomposition but generate the odd and even inverse transformed data values in a row between the even start and odd end filters of equations 88 and 89.

- Using the above even start, odd end, odd interleaved and even interleaved inverse transform digital filters, a frame rate of approximately 15 frames/second is realizable executing on a Macintosh Quadra personal computer having a 68040 microprocessor. Digital filters using the
- 10 coefficients b=5/8, a=3/8, c=1/8 and d=1/8 may also be realized in dedicated digital hardware to reduce the cost of a dedicated hardware implementation where a slightly lower compression ratio is acceptable.

To further increase software decompression speed when 15 decompressing video on a digital computer, only two octaves of inverse transform are performed on video which was previously compressed using three octaves of forward transform. This results in the low pass component of the octave 0 decomposition. The low pass component of the

- 20 octave 0 decomposition is a non-aliased high quality quarter size decimated version of the original image.

 Rather than performing octave 0 of inverse transform, horizontal linear interpolation is used to expand each row of data values of the low pass component of the octave 0
- 25 decomposition into twice the number of data values. To expand the number of rows, each row of interpolated data values is replicated once so that the total number of rows is doubled. In some embodiments, interpolation techniques other than linear interpolation are used to improve image 30 quality. For example, spline interpolation or polynomial interpolation may be used.

To further increase software execution speed when decompressing video, luminance data values are decompressed using the digital filters of equations 88, 89, 90 and 91.

35 The chrominance data values, on the other hand, are decompressed using even and odd interleaved reconstruction filters having a fewer number of coefficients than four.

In one embodiments, two coefficient odd interleaved Haar and even interleaved Haar filters are used. The even interleaved Haar reconstruction filter is:

$$D_0 = (H_0 + G_0)$$
 (equ. 92)

5 The odd interleaved Haar reconstruction filter is:

$$D_1 = (H_0 - G_0)$$
 (equ. 93)

Because the above Haar filters each only have two coefficients, there is no boundary problem as is addressed in connection with an above-described method. Accordingly, 10 another start inverse transform digital filter and another end inverse transform digital filter are not used.

To increase software execution speed still further when decompressing video, variable-length SEND and STILL_SEND tokens are used. Data values are encoded using a Huffman code as disclosed above whereas tokens are generated in variable-length form and appear in this variable-length form in the compressed data stream. This allows decompression to be performed without first calculating flags.

Figure 44 shows variable-length tokens used for encoding and decoding in accordance with some embodiments of the present invention. Because transitions from SEND mode to STOP mode or from STILL_SEND mode to STOP mode occur most frequently of the transitions indicated in 25 Figure 44, the corresponding tokens consist of only one bit.

In general, if an area changes from white to black in two consecutive frames of a video sequence and if the encoder is in LPF_SEND mode, then the difference between 30 the corresponding data values after quantization will be much larger than 37. 37 is the maximum number encodable using the specific Huffman code set forth in connection with an above-described method. Because such a large

change in data value cannot be encoded, an artifact will be generated in the decompressed image for any change in quantized data values exceeding 37. Accordingly, the Huffman code in the table below is used in accordance with 5 one embodiment of the present invention.

	HUFFMAN CODE	qindex	
	0	0	
	1s1	±1	
	1s01	±2	
10	1s001	±3	
	1s0001	±4	
	1s00001	±5	
	1s000001	±6	
	180000001	±7	
15	ls0000000 (qindex -8)	±8 ±135	

Table 6

In Table 6 above, the value (|qindex| - 8) is seven bits in length. The s in Table 6 above is a sign bit.

This embodiment is not limited to video mail
20 applications and is not limited to systems using dedicated
hardware to compress and software executing on a digital
computer to decompress. Digital circuitry of a general
purpose digital computer having a microprocessor may be
used to decode and inverse transform a compressed image

25 data stream. The coefficients 5/8, 3/8, 1/8 and 1/8 independent of sign may be the four coefficients of four coefficient high and low pass forward transform perfect reconstruction digital filters used to transform image data values into a sub-band decomposition.

WO 94/23385 PCT/GB94/00677

Although the present invention has been described by way of the above described specific embodiments, it will be understood that certain adaptations, modifications, rearrangements and combinations of various features of the 5 specific embodiments may be practiced without departing from the scope of the invention. Filters other than the four coefficient quasi-Daubechies filters can be used. In some embodiments, six coefficient quasi-Daubechies filters are used. Embodiments of this invention may, for example, 10 be practiced using a one-dimensional tree structure, a twodimensional tree structure, or a three-dimensional tree structure. Rather than testing whether or not a two-by-two block of data values is interesting, blocks of other sizes may be used. Three-by-three blocks of data values may, for 15 example, be tested. Blocks of different sizes may be used in different octaves of a decomposition. In certain embodiments, there are different types of interesting blocks. The use of tokens in combination with use of a tree structure of a decomposition to reduce the number of 20 data values encoded may be extended to include other tokens having other meanings. The "interesting with motion" token is but one example. Tree structures may be used in numerous ways to estimate the activity of a frame for rate control purposes. Numerous boundary filters, thresholds, 25 encoder and decoder modes, token schemes, tree traversing address generators, quantization schemes, Huffman-like codes, and rate control schemes will be apparent from the specific embodiments. The above-described specific embodiments are therefore described for instructional 30 purposes only and are not intended to limit the invention as set forth in the appended claims.

DATA COMPRESSION AND DECOMPRESSION GREGORY KNOWLES AND ADRIAN S. LEWIS M-2357 US APPENDIX A

```
source/Bits.c
```

```
/*
       Reading and writing bits from a file
*/
              "../include/xwave.h"
#include
#include
              "../include/Bits.h"
       bopen(name, mode)
Bits
String name, mode;
{
              bits = (Bits)MALLOC(sizeof(BitsRec));
       Bits
       if((bits-> fp=fopen(name,mode)) = = (FILE*)0)Eprintf("Failed to open binary
file\n");
             /*change*/
       bits->bufsize=0;
                            /*new*/
      bits->buf=(unsigned char)0;
                                          /*new*/
       return(bits);
}
       bclose(bits)
void
Bits
       bits;
{
      if(fclose(bits->fp)!=0) Eprintf("Failed to close binary file\n"); /*was:
fclose(bits->fp)*/
```

```
XtFree(bits);
}
       bread(bytes,num,bits)
void
unsigned char
                     *bytes;
int
       num;
Bits
       bits:
{
              byte=0, bit=0,pull,b;
       int
       bytes[byte] = 0;
       while(num > 0) {
              if (bits-> bufsize = =0) {
                     pull = fgetc(bits->fp);
                     if(pull = EOF)
                            /*printf("EOF\n"); Previously didn't check for
EOF:bits->buf=(unsigned char)fgetc(bits->fp)*/
                            for(b=byte+1;b<num/8+1;b++)
                                   bytes[b] = (unsigned char)0;
                            return;
                            }
                    bits->buf=(unsigned char)pull;
                    bits->bufsize=8;
             }
bytes[byte] = ((1\&bits->buf)!=0)?bytes[byte] | (1 < < bit):bytes[byte] \& -(1 < < bit);
             if (bit = 7) { bit = 0; byte + +; bytes[byte] = 0; }
                                                                      /* was bit = = 8 */
             else bit++;
             bits->buf=bits->buf>>1;
```

```
bits->bufsize--;
              num--;
       }
}
void bwrite(bytes,num.bits)
unsigned char
                     *bytes;
       num;
int
Bits
       bits:
{
             byte = 0, bit = 0;
       int
       unsigned char
                            xfer;
       while(num > 0) {
              if (bit = = 0) {
                     xfer=bytes[byte++];
```

```
source/Color.c
      Color routines
*/
#include
             "../include/xwave.h"
#define
             GAMMA
                           1.0/2.2
int
Visual Class \cite{Color, True Color, Static Color, Gray Scale, Static Gray}; \\
/*
      Function Name:
                          Range
      Description: Range convert for RGB/YUV calculations
      Arguments:
                   old_x - old value (0..old_r-1)
                          old_r - old range < new r
                          new_r - new range
      Returns:
                    old x scaled up to new range
*/
int
      Range(old_x,old_r,new_r)
      old_x, old_r, new_r;
int
{
      return((old_x*new_r)/old_r);
}
      Function Name:
                          Gamma
/*
      Description: Range convert with Gamma correction for RGB/YUV calculations
       Arguments:
                    as Range +
                          factor - gamma correction factor
```

```
Returns:
                     old x gamma corrected and scaled up to new range
 */
int
       Gamma(old_x,old_r,new_r,factor)
int
       old_x, old_r, new_r;
double
              factor:
{
       return((int)((double)new_r*pow((double)old_x/(double)old_r,factor)));
}
       Function Name:
                             Dither
       Description: Range convert with dithering for RGB/YUV calculations
       Arguments:
                     levels - output range (0..levels-1)
                             pixel - pixel value (0..1 < 8 + precision-1)
                             x, y - dither location
                             precision - pixel range (0..1 < 8 + precision-1)
                     dithered value (0..levels-1)
       Returns:
*/
       Dither(levels, pixel, x, y, precision)
int
int
       pixel, levels, x, y, precision;
{
              bits = 8 + precision,
       int
                     pixlev=pixel*levels,
value = (pixlev > bits) + ((pixlev-(pixlev&(-1 < bits))) > precision > global- > dither[x]
&15][y&15]?1:0);
```

```
return(value > = levels?levels-1:value);
}
/*
      Function Name:
                           ColCvt
      Description: Converts between RGB and YUV triples
      Arguments:
                    src - source triple
                           dst - destination triple
                           rgb_yuv - convert direction RGB-> YUV True
                           max - range of data (max-1..-max)
      Returns:
                    alters dst.
*/
      ColCvt(src,dst,rgb_yuv,max)
short src[3], dst[3];
Boolean
             rgb_yuv;
int
      max;
{
                    rgb_yuv_mat[2][3][3] = {{
      double
             {0.299,0.587,0.114},
             {-0.169,-0.3316,0.5},
             {0.5,-0.4186,-0.0813}
      },{
             {1,0,1.4021},
             {1,-0.3441,-0.7142},
             {1,1.7718,0}
      }};
             i, channel;
       int
       for(channel=0;channel<3;channel++) {
```

```
double
                            sum = 0.0;
              for(i=0; i < 3; i++)
sum + = (double)(src[i])*rgb yuv mat[rgb yuv?0:1][channel][i];
              dst[channel] = (int)sum < -max?-max:(int)sum > max-1?max-1:(short)sum;
       }
}
       Function Name:
                            CompositePixel
       Description: Calculates pixel value from components
       Arguments:
                     frame - Frame to be drawn on
                            x, y - coordinate of pixel in data
                            X, Y - coordinate of pixel in display
                     pixel value in colormap
       Returns:
 */
       CompositePixel(frame,x,y,X,Y)
int
Frame frame:
       x, y, X, Y;
int
{
       Video vid=frame-> video;
              channel=frame->channel, pixel, value=0;
       int
       if (channel! = 3) {
pixel = (int)vid - > data[channel][frame - > frame][Address2(vid, channel, x, y)] + (128 < < vid-
> precision);
              value = Dither(global-> levels, pixel, X, Y, vid-> precision);
       } else for(channel=0;channel<3;channel++) {
              int
```

```
levels = vid-> type = = RGB?global-> rgb_levels:global-> yuv_levels[channel];
pixel = (int)vid->data[channel][frame-> frame][Address(vid,channel,x,y)]+(128 < < vid-
> precision),
              value = levels*value + Dither(levels, pixel, X, Y, vid-> precision);
       }
       return(value);
}
      InitVisual()
void
{
                     *dpy = XtDisplay(global-> toplevel);
       Display
       int
              scrn=XDefaultScreen(dpy), class=0, depth=8, map, i, r, g, b, y, u, v;
       String
VisualNames[6] = {"PseudoColor", "DirectColor", "TrueColor", "StaticColor", "GrayScale",
"StaticGray"};
       XColor
                     color:
       global-> visinfo = (XVisualInfo *)MALLOC(sizeof(XVisualInfo));
       while(depth > 0
&&!XMatchVisualInfo(dpy,scm,depth,VisualClass[class],global->visinfo))
              if (class = = 5) {class = 0; depth--;} else class + +;
       Dprintf("Visual: %s depth %d\n", VisualNames[class], depth);
       global->palettes=(Palette)MALLOC(sizeof(PaletteRec));
       strcpy(global-> palettes-> name, "Normal");
       global->palettes->next=NULL;
       global > no pals = 1;
       switch(global->visinfo->class) {
       case TrueColor:
       case DirectColor:
```

```
case StaticColor:
      case GrayScale:
             fprintf(stderr, "Unsupported visual type: %s\n", VisualNames[class]);
             exit();
             break:
      case PseudoColor:
             global->levels=global->visinfo->colormap_size;
             global->rgb_levels=(int)pow((double)global->levels,1.0/3.0);
             for(map=0;map<2;map++) { /* rgb non-gamma and gamma maps */
global->cmaps[map] = XCreateColormap(dpy, XDefaultRootWindow(dpy), global->visinfo
-> visual, Alloc All);
                    for(r=0;r < global > rgb levels;r++)
                          for(g=0;g < global > rgb_levels;g++)
                                 for(b=0;b < global > rgb_levels;b++) {
color.pixel=(r*global->rgb_levels+g)*global->rgb_levels+b;
color.red = (map&1)?Gamma(r,global->rgb_levels,65536,GAMMA):Range(r,global->rg
b levels,65536);
color.green = (map&1)?Gamma(g,global-> rgb_levels,65536,GAMMA):Range(g,global->
rgb_levels,65536);
color.blue = (map&1)?Gamma(b,global->rgb_levels,65536,GAMMA):Range(b,global->r
gb levels, 65536);
                                       color.flags=DoRed | DoGreen | DoBlue;
XStoreColor(dpy,global->cmaps[map],&color);
                   color.pixel = global-> levels-1;
                   color.red = 255 < < 8;
```

```
color.green=255 < < 8;
                    color.blue = 255 < < 8;
                    color.flags = DoRed | DoGreen | DoBlue;
                    XStoreColor(dpy,global->cmaps[map],&color):
             }
             for(map=2;map<4;map++) { /* mono non-gamma and gamma maps */
global->cmaps[map] = XCreateColormap(dpy, XDefaultRootWindow(dpy), global->visinfo
-> visual, AllocAll);
                    for(i=0;i < global > visinfo > colormap_size;i++) {
                           color.pixel=i;
color.red=(map&1)?Gamma(i,global->levels,65536,GAMMA):Range(i,global->levels,6
5536);
color.green=(map&1)?Gamma(i,global-> levels,65536,GAMMA):Range(i,global-> levels
,65536);
color.blue = (map&1)?Gamma(i,global-> levels,65536,GAMMA):Range(i,global-> levels.
65536);
                          color.flags = DoRed | DoGreen | DoBlue;
                          XStoreColor(dpy,global->cmaps[map],&color);
                    }
             }
             global - yuv_levels[0] = (int)pow((double)global - > levels, 1.0/2.0);
             global->yuv levels[1] = (int)pow((double)global-> levels, 1.0/4.0);
             global > yuv levels[2] = (int)pow((double)global > levels, 1.0/4.0);
             for(map=4;map<6;map++) { /* yuv non-gamma and gamma maps */
global->cmaps[map] = XCreateColormap(dpy, XDefaultRootWindow(dpy), global-> visinfo
-> visual, AllocAll);
                    for(y=0;y < global-> yuv_levels[0];y++)
```

```
- 100 -
```

```
for(u=0;u < global-> yuv_levels[1];u++)
                                                                                                                                                                                             for(v=0; v < global-> yuv levels[2]; v++) {
                                                                                                                                                                                                                                  short
src[3] = \{(short)(Range(y,global->yuv levels[0],65536)-32768),
(\text{short})(\text{Range}(u,\text{global-}>\text{yuv levels}[1],65536)-32768),
(\text{short})(\text{Range}(v,\text{global-}>\text{yuv levels}[2],65536)-32768)\}, dst[3];
                                                                                                                                                                                                                                 ColCvt(src,dst,False,65536/2);
 color.pixel = (y*global->yuv_levels[1]+u)*global->yuv_levels[2]+v;
color.red = (map&1)?Gamma((int)dst[0] + 32768,65536,65536,GAMMA):(int)dst[0] + 32768,65536,GAMMA)
 8;
color.green = (map&1)?Gamma((int)dst[1] + 32768,65536,65536,GAMMA):(int)dst[1] + 32768,65536,GAMMA):(int)dst[1] + 32768,GAMMA):(int)dst[1] + 32768,GAMMA]:(int)dst[1] + 32768,GAMMA]:(int)d
 768;
color.blue = (map&1)?Gamma((int)dst[2] + 32768,65536,65536,GAMMA):(int)dst[2] + 32768,65536,GAMMA):(int)dst[2] + 32768,GAMMA):(int)dst[2] + 32768,GAMMA]:(int)dst[2] + 32768,GAMMA]:(int)ds
68;
                                                                                                                                                                                                                               color.flags=DoRed | DoGreen | DoBlue;
XStoreColor(dpy,giobal->cmaps[map],&color);
                                                                                                              color.pixel = global-> levels-1;
                                                                                                               color.red = 255 < < 8:
                                                                                                              color.green = 255 < < 8;
                                                                                                              color.blue = 255 < < 8;
                                                                                                              color.flags=DoRed | DoGreen | DoBlue;
                                                                                                              XStoreColor(dpy,global->cmaps[map],&color);
                                                                         }
```

PCT/GB94/00677

```
global-> palettes-> mappings = NULL;
             break:
      case StaticGray:
             global - > levels = 1 < < depth;
             for(i=0; i<6; i++) global-> cmaps[i] = XDefaultColormap(dpy,scrn);
             color.pixel=0;
             XQueryColor(dpy,XDefaultColormap(dpy,scrn),&color);
             if (color.red = = 0 \&\& color.green = = 0 \&\& color.blue = = 0)
global->palettes->mappings=NULL;
             else {
                    global->palettes->mappings=(Map)MALLOC(sizeof(MapRec));
                    global-> palettes-> mappings-> start = 0;
                    global-> palettes-> mappings-> finish = global-> levels-1;
                    global-> palettes-> mappings-> m=-1;
                    global->palettes->mappings->c=global->levels-1;
                    global-> palettes-> mappings-> next = NULL;
             }
             break;
       }
}
             ChannelCmap(channel,type,gamma)
Colormap
       channel;
int
VideoFormat type;
Boolean
             gamma;
       Colormap
                    cmap;
       if (channel!=3 \mid | type==MONO) 
              if (gamma) cmap=global->cmaps[global->cmaps[2] == NULL?3:2];
```

```
- 102 -
```

```
else cmap=global->cmaps[global->cmaps[3] == NULL?2:3];
} else if (type==RGB) {
    if (gamma) cmap=global->cmaps[global->cmaps[0] == NULL?1:0];
    else cmap=global->cmaps[global->cmaps[1] == NULL?0:1];
} else {
    if (gamma) cmap=global->cmaps[global->cmaps[4] == NULL?5:4];
    else cmap=global->cmaps[global->cmaps[5] == NULL?4:5];
}
return(cmap);
}
```

source/Convert.c

```
"../include/xwave.h"
#include
short cti(c)
char
       c;
{
       return((short)(c)^-128);
}
char
       itc(i)
short i;
{
       static int
                     errors=0;
      if (i<-128 || i>127) {
              if (errors = = 99) {
                     Dprintf("100 Conversion overflows\n");
                     errors=0;
             } else errors++;
             i=(i<-128)?-128:127;
      return((char)(i^128));
}
```

source/Convolve3.c

```
/*
       2D wavelet transform convolver (fast hardware emulation)
       New improved wavelet coeffs: 11 19 5 3
*/
              "../include/xwave.h"
#include
       Function Name:
                            Round
       Description: Rounding to a fixed number of bits, magnitude rounded down
       Arguments:
                    number - number to be rounded
                            bits - shifted bits lost from number
       Returns: rounded number
 */
short Round(number, bits)
int
       number:
int
       bits;
{
       if (bits = =0) return((short)number);
       else return((short)(number + (1 < bits-1)-(number < 0?0:1) >> bits));
}
/*
       Function Name:
                           Convolve
       Description: Perform a wavelet convolution on image data
       Arguments:
                    data - data to be transformed
                           dirn - convolution direction
```

..........

```
size - size of image data
                              oct src, oct dst - initial and final octave numbers
       Returns:
                      data altered
 */
void
       Convolve(data, dirn, size, oct_src, oct_dst)
short *data;
Boolean
               dirn;
       size[2], oct_src, oct_dst;
int
{
       int
               tab[4][4], addr[4] = \{-1,-1,-1,-1\}, index, mode, i, j, oct, orient,
area = size[0]*size[1];
       Boolean
                      fwd rev=oct src<oct dst;
               windows[12][5] = {
       int
              \{1,2,3,-4,2\}, /* 0 - normal forward 0 */
              \{4,-3,2,1,3\}, /* 1 - normal forward 1 */
              \{1,-2,3,4,2\}, /* 2 - normal reverse 0 */
              \{4,3,2,-1,3\}, /* 3 - normal reverse 1 */
              \{2,3,4,-4,3\}, /* 4 - end forward 0 */
              \{4,-4,3,2,4\}, /* 5 - end forward 1 */
              \{2,2,3,-4,2\}, /* 6 - start forward 0 */
              {4,-3,2,2,3}, /* 7 - start forward 1 */
              \{3,-4,-4,3,4\}, /* 8 - break reverse end dirn = = False*/
              \{4,3,-3,-4,3\}, /* 9 - break reverse start dirn==False */
              \{-3,-4,4,3,4\}, /* 10 - break reverse end dim = = True */
              \{-4,3,3,-4,3\}, /* 11 - break reverse start dirn = = True */
       }, win[3];
                                     /* 12 - no calculation */
       for(oct=oct_src;oct!=oct_dst;oct+=(fwd_rev?1:-1)) {
            long shift = oct-(fwd_rev?0:1);
```

A MATTER TO STORE /13 8 P AA

```
for(orient=0;orient<2;orient++) {
       Boolean
                      x y = fwd rev = = (orient = = 0);
for (index = 0; index < (area > > (shift < < 1)); index + +) {
              major, minor, value, valuex3, valuex11, valuex19, valuex5;
       major = index/(size[x_y?0:1] > shift);
       minor = index-major*(size[x_y?0:1] > > shift);
       for(j=0; j<3; j++) win[j]=12;
       switch(minor) {
       case 0: break;
       case 1: if (!fwd_rev) win[0] = dirn?11:9; break;
       case 2: if (fwd rev) \{ win[0]=6; win[1]=7; \}; break:
       default:
                      if (minor+1 = size[x y?0:1] > shift)
                             if (fwd_rev) { win[0]=4; win[1]=5; }
                             else { win[0] = 2; win[1] = 3; win[2] = dirn?10:8; }
                     } else if (fwd rev) {
                             if ((1\&minor) = =0) \{ win[0] = 0; win[1] = 1; \}
                     } clse {
                             if ((1\&\min_{0 \le i \le 1} e^{-1}) \{ win[0] = 2; win[1] = 3; \}
                     }
      addr[3\&index] = (x_y?minor:major) + size[0]*(x_y?major:minor) < shift;
      value = (int)data[addr[3&index]];
      valuex5 = value + (value < < 2);
      valuex3 = value + (value < < 1);
      valuex11 = valuex3 + (value < < 3):
      valuex19 = valuex3 + (value < < 4):
      tab[3&index][3] = fwd rev | | !dirn?valuex3:valuex19;
      tab[3&index][2] = fwd rev || dirn?valuex5:valuex11;
```

```
source/Copy.c
```

```
/*
        Copy video, includes direct copy, differencing, LPF zero, LPF only, RGB-YUV
 conversion and gamma correction
 */
 #include
               "../include/xwave.h"
 #include
               "Copy.h"
 extern int
               Shift();
               ColCvt();
 extern void
        CopyVideoCtrl(w,closure,call_data)
 void
Widget
               w:
 caddr t closure, call data;
 {
        CopyCtrl
                     ctrl = (CopyCtrl)closure;
        Video new=CopyHeader(ctrl-> video), src=ctrl-> video;
               frame, channel, i, x, y, X, Y, map[256];
        int
        if (global-> batch = = NULL)
 ctrl-> mode = (int)XawToggleGetCurrent(ctrl-> radioGroup);
        strcpy(new-> name,ctrl-> name);
        strcpy(new-> files, new-> name);
        switch(ctrl-> mode) {
              1:
                     Dprintf("Direct copy\n");
        case
                            new-> UVsample[0] = ctrl-> UVsample[0];
                            new-> UVsample[1] = ctrl-> UVsample[1];
```

```
break:
                    Dprintf("Differences\n");
             2:
      case
                           break;
                    Dprintf("LPF zero\n");
             3:
      case
                           break:
                    Dprintf("LPF only\n");
             4:
      case
                           new-> trans.type = TRANS_None;
new-> size[0] = new-> size[0] > new-> trans. wavelet. space[0];
new-> size[1] = new-> size[1] > new-> trans. wavelet.space[0];
                           break:
                    Dprintf("RGB-YUV\n");
             5:
      case
                           new > type = new > type = YUV?RGB:YUV
                           new-> UVsample[0] = 0;
                           new-> UVsample[1]=0;
                          break:
                    Dprintf("Gamma conversion\n");
             6:
      case
                          new->gamma=!new->gamma;
                          for(i=0; i < 256; i++)
map[i] = gamma(i, 256, new-> gamma?0.5:2.0);
                          break;
      }
      if (new->disk==True) SaveHeader(new);
      for(frame = 0; frame < new-> size[2]; frame + +) {
             GetFrame(src,frame);
             NewFrame(new,frame);
             switch(ctrl-> mode) {
                  1:
             case
for(channel = 0; channel < (new-> type = = MONO?1:3); channel + +) {
                                        int
                                               size = Size(new,channel,0)*Size(new,channel,1);
```

```
- 110 -
                                          for (y = 0; y < Size(new, channel, 1); y + +)
                                                 for(x = 0; x < Size(new, channel, 0); x + +)
new-> data[channel][frame][x+Size(new,channel,0)*y] = src-> data[channel][frame][Shift(
x,src->type = = YUV &&
channel! = 0?new-> UVsample[0]-src-> UVsample[0]:0) + Size(src,channel,0) * Shift(y,src-
> type = = YUV && channel! = 0?new-> UVsample[1]-src-> UVsample[1]:0)];
                                   break:
                     2:
              case
for(channel = 0; channel < (new-> type = = MONO?1:3); channel + +) {
size = Size(new,channel,0)*Size(new,channel,1);
                                          for(i=0; i < size; i++)
new-> data[channel][i] = src-> data[channel][frame][i]-(frame = = 0?0:src-> data[channel][frame][i]
annel][frame-1][i]);
                                          }
                                   break:
              case
                     3:
for(channel=0;channel<(new->type==MONO?1:3);channel++) {
                                          int
size = Size(new,channel,0)*Size(new,channel,1);
                                          for(i=0;i < size;i++) {
                                                 x = i\%Size(new, channel, 0);
y = i/Size(new, channel, 0);
                                                 if
(x\%(1 < new-> trans. wavelet.space[new-> type==YUV && channel!=0?1:0])==0
&& y\%(1 < new-> trans.wavelet.space[new-> type = = YUV &&
channel! = 0?1:0) = = 0)
```

```
- 111 -
new->data[channel][frame][i] = 0;
                                                   else
new->data[channel][frame][i] = src->data[channel][frame][i];
                                    break;
                      4:
              case
for(channel = 0; channel < (new-> type = = MONO?1:3); channel + +) {
                                            int
size = Size(new,channel,0)*Size(new,channel,1);
                                           for(i=0; i < size; i++) {
                                                   x = i\%Size(new, channel, 0);
y = i/Size(new, channel, 0);
new-> data[channel][frame][i] = src-> data[channel][frame][(x+(y<< new-> trans. wavele
t.space[0])*Size(new,channel,0)) < < new-> trans.wavelet.space[0]];
                                    break;
                             for(X=0;X < new > size[0];X++)
              case
for(Y=0;Y < new-> size[1];Y++) {
                                           short src_triple[3], dst_triple[3];
                                           for(channel = 0; channel < 3; channel + +)
src triple[channel] = src-> data[channel][frame][Address(src, channel, X, Y)];
ColCvt(src_triple,dst_triple,new->type = = YUV,1 < <7+new->precision);
                                           for(channel=0; channel<3; channel++)
                                                  new->data[dante][frame][Addes(new_dantel_X,Y)]=ds_triple[dantel];
                                    }
```

SUBSTITUTE SHEET (RULE 26)

```
break:
                     6:
              case
for(channel = 0; channel < (new-> type = = MONO?1:3); channel + +) {
size = Size(new,channel.0)*Size(new,channel.1);
                                           for(i=0; i < size; i++)
new->data[channel][frame][i] = map[src->data[channel][frame][i] + 128]-128;
                                   break;
                     }
              if (frame > 0) FreeFrame(src, frame-1);
              SaveFrame(new,frame);
              FreeFrame(new,frame);
       }
       FreeFrame(src, src-> size[2]-1);
       new->next=global->videos;
       global-> videos = new;
}
void
       BatchCopyCtrl(w,closure,call_data)
Widget
caddr t
             closure, cali data;
{
                     ctrl=(CopyCtrl)closure;
       CopyCtrl
       if (ctrl-> video = = NULL)
ctrl-> video = FindVideo(ctrl-> src_name, global-> videos);
      CopyVideoCtrl(w,closure,call data);
}
```

```
CopyCtrl
             InitCopyCtrl(name)
String name;
{
                    ctrl = (CopyCtrl)MALLOC(sizeof(CopyCtrlRec));
      CopyCtrl
      strcpy(ctrl-> src_name,name);
      strcpy(ctrl->name,name);
      ctrl-> mode=1;
      return(ctrl);
}
#define
             COPY_ICONS
                                  17
     CopyVideo(w,closure,call_data)
Widget
             w;
caddr t
             closure, call_data;
{
      Video video = (Video) closure;
      CopyCtrl
                    ctrl = InitCopyCtrl(video-> name);
      NumInput
                    UVinputs = (NumInput)MALLOC(2*sizeof(NumInputRec));
                    msg = NewMessage(ctrl-> name, NAME_LEN);
      Message
                          destroy_call[] = {
      XtCallbackRec
             {Free,(caddr_t)ctrl},
             {Free,(caddr_t)UVinputs},
             {CloseMessage,(caddr_t)msg},
             {NULL, NULL},
      };
      Widget
                    shell = ShellWidget("copy_video", w, SW_below, NULL, destroy call),
```

```
form = FormatWidget("cpy form", shell), widgets[COPY ICONS]:
      FormItem
                    items\Pi = \{
             {"cpy cancel", "cancel", 0, 0, FW icon, NULL}.
             {"cpy_confirm", "confirm", 1,0,FW icon, NULL},
             {"cpy title", "Copy a video", 2,0,FW label, NULL},
             {"cpy vid lab", "Video Name: ",0,3,FW label, NULL}.
             {"cpy text", NULL, 4, 3, FW text, (String) msg},
             {"cpy_copy", "copy", 0,5,FW_toggie, NULL},
             {"cpy diff", "diff", 6,5,FW toggle, (String)6},
             {"cpy_lpf_zero","lpf_zero",7,5,FW_toggle,(String)7},
             {"cpy_lpf_only","lpf_only",8,5,FW_toggle,(String)8},
             {"cpy_color", "color_space", 9, 5, FW_toggle, (String)9},
             {"cpy_gamma", "gamma", 10,5,FW toggle, (String) 10},
             {"cpy_UV0_int",NULL,0,6,FW_integer,(String)&UVinputs[0]},
             {"cpy_UV0_down", NULL, 12,6,FW_down, (String)&UVinputs[0]},
             {"cpy_UV0_up", NULL, 13,6,FW_up,(String)&UVinputs[0]},
             {"cpy_UV1_int", NULL, 0, 14, FW_integer, (String) & UV inputs[1]},
             {"cpy UV1 down", NULL, 12, 14, FW down, (String) & UV inputs[1]},
             {"cpy_UV1_up",NULL,16,14,FW_up,(String)&UVinputs[1]},
      };
      XtCallbackRec
                          callbacks[]={
             {Destroy,(caddr_t)shell},
             {NULL, NULL},
             {CopyVideoCtrl,(caddr t)ctrl},
             {Destroy,(caddr t)shell},
             {NULL, NULL}.
             {NULL, NULL}, {NULL, NULL}, {NULL, NULL}, {NULL, NULL},
{NULL, NULL}, {NULL, NULL},
             {NumIncDec,(caddr t)&UVinputs[0]}, {NULL,NULL},
```

};

```
{NumIncDec,(caddr_t)&UVinputs[0]}, {NULL,NULL},
             {NumIncDec,(caddr_t)&UVinputs[1]}, {NULL,NULL}.
             {NumIncDec,(caddr_t)&UVinputs[1]}, {NULL,NULL},
      };
      Dprintf("CopyVideo\n");
      msg->rows=1; msg->cols=NAME LEN;
      ctrl-> video = video:
      UVinputs[0].format = "UV sub-sample X: %d";
      UVinputs[0].min=0;
      UVinputs[0].max = 2;
      UVinputs[0].value = &ctrl-> UVsample[0]:
      UVinputs[1].format = "UV sub-sample Y: %d":
      UVinputs[1].min=0;
      UVinputs[1].max = 2;
      UVinputs[1].value = &ctrl-> UVsample[1];
      ctrl-> UVsample[0] = video-> UVsample[0];
      ctrl-> UVsample[1] = video-> UVsample[1];
      FillForm(form, COPY_ICONS, items, widgets, callbacks);
      ctrl-> radioGroup = widgets[5];
      XtSetSensitive(widgets[6], video-> size[2] > 1);
      XtSetSensitive(widgets[7], video-> trans.type! = TRANS_None);
      XtSetSensitive(widgets[8], video-> trans.type! = TRANS_None);
      XtSetSensitive(widgets[9], video-> type! = MONO);
      XtSetSensitive(widgets[10], video-> type! = YUV &&
video-> trans.type = = TRANS None);
      XtPopup(shell, XtGrabExclusive);
```

source/Frame.c

```
/*
      Frame callback routines for Destroy
*/
             "../include/xwave.h"
#include
#include
              <X11/Xmu/SysUtil.h>
#include
              <pwd.h>
extern void
             CvtIndex();
extern Palette
                    FindPalette();
extern void
             SetSensitive();
typedef
             struct {
      Frame frame:
             frame_number, frame_zoom, frame_palette, frame_channel;
} ExamCtrlRec, *ExamCtrl;
      FrameDestroy(w,closure,call_data)
Widget
             w;
caddr t
             closure, call_data;
{
      Frame frame = (Frame)closure;
      void CleanUpPoints(), FrameDelete();
      Dprintf("FrameDestroy\n");
      frame->point->usage--;
       if (frame->msg!=NULL) {
```

```
frame->msg->shell=NULL;
             CloseMessage(NULL,(caddr_t)frame->msg,NULL);
      }
      if (frame->point->usage==0) CleanUpPoints(&global->points);
      XtPopdown(frame-> shell);
      XtDestroyWidget(frame-> shell);
      FrameDelete(&global-> frames, frame);
}
      CleanUpPoints(points)
void
Point *points;
{
      Point dummy = *points;
      if (dummy! = NULL) {
             if (dummy->usage<1) {
                    *points = dummy-> next;
                   XtFree(dummy);
                   CleanUpPoints(points);
             } else CleanUpPoints(&((*points)->next));
      };
}
      FrameDelete(frames, frame)
void
Frame *frames, frame;
{
      if
             (*frames! = NULL) {
             if (*frames = = frame) {
```

```
int
                            number = frame- > frame:
                     frame \rightarrow frame = -1;
                     FreeFrame(frame-> video, number);
                     *frames = frame- > next;
                     XtFree(frame);
              } else FrameDelete(&(*frames)->next.frame);
       }
}
       ExamineCtrl(w,closure,call data)
Widget
caddr t
              closure, call_data;
       ExamCtrl
                     ctrl=(ExamCtrl)closure;
       Arg
              args[1];
       if (ctrl-> frame-> frame! = ctrl-> frame_number-ctrl-> frame-> video-> start) {
              int
                     old_frame=ctrl->frame->frame;
             ctrl-> frame-> frame = ctrl-> frame_mumber-ctrl-> frame-> video-> start;
              FreeFrame(ctrl->frame-> video,old frame);
              GetFrame(ctrl-> frame-> video,ctrl-> frame-> frame);
      ctrl-> frame-> zoom = ctrl-> frame_zoom;
      ctrl-> frame-> palette = ctrl-> frame_palette;
      ctrl-> frame-> channel = ctrl-> frame_channel;
      XtSetArg(args[0], XtNbitmap, UpdateImage(ctrl-> frame));
      XtSetValues(ctrl-> frame-> image_widget, args, ONE);
```

```
XtSetArg(args[0], XtNcolormap, ChannelCmap(ctrl-> frame-> channel, ctrl-> frame-> vide
o->type,ctrl->frame->video->gamma));
      XtSetValues(ctrl-> frame-> shell, args, ONE);
      if (ctrl-> frame-> msg! = NULL) UpdateInfo(ctrl-> frame);
}
#define
             EXAM ICONS
                                  13
      Examine(w,closure,call data)
Widget
             w;
caddr t
             closure, call data;
{
      ExamCtrl
                    ctrl = (ExamCtrl)MALLOC(sizeof(ExamCtrlRec)):
                    num inputs = (NumInput)MALLOC(2*sizeof(NumInputRec));
      NumInput
      XtCallbackRec destroy call[]={
             {Free,(caddr t)ctrl},
             {Free,(caddr_t)num_inputs},
             {NULL, NULL},
      }, pal_call[2*global->no_pals];
      Widget
                    shell = ShellWidget("examine", w, SW below, NULL, destroy call).
                    form = FormatWidget("exam_form", shell), widgets[EXAM_ICONS],
                    pal_widgets[global->no_pals], pal_shell;
      Frame frame=(Frame)closure;
      FormItem
                   items[] = {
             {"exam_cancel", "cancel", 0, 0, FW_icon, NULL},
             {"exam confirm", "confirm", 1,0,FW icon, NULL},
             {"exam_label", "Examine", 2, 0, FW label, NULL},
             {"exam_ch_lab", "Channel: ",0,3,FW label,NULL},
{"exam_ch_btn", ChannelName[frame->video->type][frame->channel], 4, 3, FW_button, "
```

```
exam_cng_ch"},
             {"exam pal lab", "Palette: ",0,4,FW label, NULL},
{"exam_pal_bm", FindPalette(global-> palettes, frame-> palette)-> name, 4, 4, FW button, "
exam cng_pal"},
             {"exam z int", NULL, 0, 6, FW integer, (String) & num inputs[0]},
             {"exam z dowm", NULL, 8, 6, FW down, (String) & num inputs[0]},
             {"exam_z_up", NULL, 9, 6, FW_up, (String)&num_inputs[0]},
             {"exam zoom int", NULL, 0, 8, FW integer, (String) & num inputs[1]},
             {"exam zoom dowm", NULL, 8, 8, FW down, (String) & num inputs[1]}.
             {"exam_zoom_up", NULL, 12, 8, FW_up, (String)&num_inputs[1]},
       };
      MenuItem
                    pal_memu[global->no_pals];
      XtCallbackRec
                           callbacks[] = {
             {Destroy,(caddr t)shell},
             {NULL, NULL},
             {ExamineCtrl,(caddr_t)ctrl},
             {Destroy,(caddr_t)shell},
             {NULL, NULL},
             {NumIncDec,(caddr t)&num inputs[0]}, {NULL,NULL},
             {NumIncDec,(caddr t)&mum inputs[0]}, {NULL,NULL},
             {NumIncDec,(caddr_t)&num_inputs[1]}, {NULL,NULL},
             {NumIncDec,(caddr t)&num inputs[1]}, {NULL, NULL},
       };
       int
             i. width=0;
      Palette
                    pal = global-> palettes;
      XFontStruct *font;
             args[1];
      Arg
                           dummy[global->no pals], dummy2[global->no pals]; /*
      caddr t
gcc-mc68020 bug avoidance */
       Dprintf("Examine\n");
```

```
ctrl-> frame = frame;
ctrl-> frame_number = frame-> frame + frame-> video-> start;
ctrl-> frame zoom=frame-> zoom;
ctrl-> frame palette=frame-> palette;
ctrl-> frame channel=frame-> channel:
num_inputs[0].format = "Frame: %03d";
num_inputs[0].max = frame-> video-> start + frame-> video-> size[2]-1;
num inputs[0].min=frame-> video-> start:
num_inputs[0].value=&ctrl-> frame number;
num_inputs[1].format="Zoom: %d";
num inputs[1].max = 4;
num inputs[1].min=0;
num inputs[1].value=&ctrl-> frame zoom:
FillForm(form, EXAM_ICONS, items, widgets, callbacks);
font = FindFont(widgets[6]);
for(i=0;pal!=NULL;pal=pal->next,i++)
      pal_memu[i].name = pal- > name;
      pal_memu[i].widgetClass=smeBSBObjectClass;
      pal_menu[i].label=pal->name;
      pal memu[i].hook=NULL;
      pal call[i*2].callback=SimpleMenu:
      pal_call[i*2].closure = (caddr_t)&ctrl-> frame_palette;
      pal call[i*2+1].callback=NULL;
      pal call[i*2+1].closure=NULL;
      width=TextWidth(width,pal->name,font);
}
pal_shell=ShellWidget("exam_cng_pal",shell,SW_menu,NULL,NULL);
FillMenu(pal shell, global-> no pals, pal menu, pal widgets, pal call);
XtSetArg(args[0],XtNwidth,2+width);
XtSetValues(widgets[6], args, ONE);
```

```
if (frame-> video-> type = = MONO) XtSetSensitive(widgets[4], False);
      else {
             Menultem
                           ch menu[4];
             Widget
ch_shell=ShellWidget("exam_cng_ch",shell,SW_menu,NULL,NULL), ch_widgets[4];
             XtCallbackRec
                                  ch call[8];
             font=FindFont(widgets[4]);
             width = 0:
             for(i=0;i<4;i++) {
                    ch_menu[i].name = ChannelName[frame-> video-> type][i];
                    ch_menu[i].widgetClass = smeBSBObjectClass;
                    ch_menu[i].label=ChannelName[frame-> video-> type][i];
                    ch_menu[i].hook=(caddr_t)&ctrl-> frame channel;
                    ch call[i*2].callback=SimpleMenu;
                    ch call[i*2].closure=(caddr t)&ctrl-> frame channel;
                    ch_call[i*2+1].callback=NULL;
                    ch_call[i*2+1].closure=NULL;
width=TextWidth(width,ChannelName[frame->video->type][i],font);
             }
             FillMenu(ch_shell,4,ch_menu,ch_widgets,ch_call);
             XtSetArg(args[0], XtNwidth, 2 + width);
             XtSetValues(widgets[4], args, ONE);
      XtPopup(shell, XtGrabExclusive);
}
      FramePointYN(w,closure,call data)
void
Widget
             w;
             closure, call_data;
caddr t
```

```
{
       Frame frame = (Frame) closure;
       Arg
              args[1];
       Pixmap
                     pixmap;
       Display
                     *dpy = XtDisplay(global-> toplevel);
       Icon point_y = FindIcon("point y"),
                     point_n = FindIcon("point_n");
       Dprintf("FramePointYN\n");
       frame->point_switch=!frame->point_switch;
       XtSetSensitive(frame-> image_widget,frame-> point_switch);
       XtSetArg(args[0], XtNbitmap, (frame->point_switch?point_y:point_n)->pixmap);
       XtSetValues(w, args, ONE);
       XtSetArg(args[0], XtNbitmap, &pixmap);
       XtGetValues(frame->image_widget,args,ONE);
       UpdatePoint(dpy,frame,pixmap);
       XtSetArg(args[0], XtNbitmap, pixmap);
       XtSetValues(frame-> image_widget, args, ONE);
       if (frame-> msg! = NULL) UpdateInfo(frame);
}
rvoid
      NewPoint(w,closure,call data)
Widget
caddr t
             closure, call_data;
      Frame = (Frame)closure;
       Video vid=frame-> video;
       void UpdateFrames();
             *posn=(int *)call_data,
      ·int
channel = frame -> channel == 3?0: frame -> channel:
```

```
posn[0] = posn[0] > frame - zoom; posn[1] = posn[1] > frame - zoom;
       if (vid->trans.type = = TRANS_Wave) {
                    octs=vid->trans.wavelet.space[vid->type==YUV &&
channel! = 0.1:0], oct;
CvtIndex(posn[0],posn[1],Size(vid,channel,0),Size(vid,channel,1),octs,&posn[0],&posn[1]
.&oct):
      if (vid->type==YUV && channel!=0) {
             posn[0] = posn[0] < vid-> UVsample[0];
             posn[1] = posn[1] < vid-> UVsample[1];
      Dprintf("NewPoint %d %d previous %d
%d\n",posn[0],posn[1],frame->point->location[0],frame->point->location[1]);
      if (posn[0]!=frame->point->location[0] ||
posn[1]! = frame-> point-> location[1]) {
             UpdateFrames(global-> frames, frame-> point, False);
             frame->point->location[0] = posn[0];
             frame->point->location[1] = posn[1];
             UpdateFrames(global-> frames, frame-> point, True);
      } else Dprintf("No movement\n");
}
      UpdateFrames(frame, point, update)
void
Frame frame;
Point point;
Boolean
             update;
             args[1];
      Arg
```

```
if (frame! = NULL) {
             if (point = = frame-> point && frame-> point_switch = = True) {
                   Pixmap
                                 pixmap;
                   Display
                                 *dpy = XtDisplay(global-> toplevel);
                   XtSetArg(args[0],XtNbitmap,&pixmap);
                   XtGetValues(frame->image widget,args,ONE);
                   UpdatePoint(dpy,frame,pixmap);
                   if (update = = True) {
                          XtSetArg(args[0],XtNbitmap,pixmap);
                          XtSetValues(frame-> image_widget, args, ONE);
                          if (frame->msg!=NULL) UpdateInfo(frame);
                   }
             UpdateFrames(frame->next,point,update);
      }
}
      CloseInfo(w,closure,call_data)
Widget
             w;
caddr_t
             closure, cali_data;
{
      Frame frame = (Frame)closure;
      frame-> msg = NULL;
}
                                 2
#define
             INFO ICONS
void FrameInfo(w,closure,call_data)
```

```
Widget
             w;
             closure, call_data;
caddr_t
{
      Frame frame = (Frame)closure;
                    msg = NewMessage(NULL, 1000);
      Message
      XtCallbackRec
                           callbacks[] = {
              {SetSensitive,(caddr_t)w},
              {CloseInfo,(caddr_t)frame},
              {CloseMessage,(caddr_t)msg},
              {NULL, NULL},
       };
       Dprintf("FrameInfo\n");
       frame-> msg = msg;
       UpdateInfo(frame);
       TextSize(msg);
       MessageWindow(w,msg,frame-> video-> name,True,callbacks);
       XtSetSensitive(w,False);
}
       FrameMerge(w,closure,call_data)
void
Widget
              closure, call_data;
caddr t
 {
       Frame frame = (Frame)closure;
       void MergePoints();
        Arg
              args[1];
        Dprintf("FrameMerge\n");
        MergePoints(global-> frames, frame);
```

```
}
      MergePoints(frame_search,frame_found)
Frame frame_search, frame_found;
{
      Arg
             args[1];
      if (frame_search! = NULL) {
             if (NULL = = XawToggleGetCurrent(frame_search-> point_merge_widget)
|| frame_search = = frame_found)
                    MergePoints(frame_search-> next,frame_found);
             else {
                    Pixmap
                                 pixmap;
                                 *dpy = XtDisplay(global-> toplevel);
                    Display
                    XtSetArg(args[0], XtNbitmap, &pixmap);
                    XtGetValues(frame_found->image_widget,args,ONE);
                    if (frame_found->point_switch = = True)
UpdatePoint(dpy,frame_found,pixmap);
                    frame_search->point->usage++;
                    frame_found->point->usage--;
                    if (frame_found->point->usage==0)
CleanUpPoints(&global->points);
                    frame_found->point=frame_search->point;
                    if (frame_found->point_switch = = True) {
                           UpdatePoint(dpy,frame_found,pixmap);
                           XtSetArg(args[0], XtNbitmap, pixmap);
                           XtSetValues(frame found->image_widget,args,ONE);
                    if (frame found->msg!=NULL) UpdateInfo(frame_found);
```

```
XawToggleUnsetCurrent(frame_search->point_merge_widget);
                    XawToggleUnsetCurrent(frame_found->point_merge_widget);
      }
}
#define
             POST DIR
                            "postscript"
void
      PostScript(w,closure,call_data)
Widget
             w;
caddr t
             closure, call data;
{
      Frame frame=(Frame)closure;
      Video video=frame-> video;
      FILE *fp, *fopen();
             file name[STRLEN], hosmame[STRLEN];
      char
             x, y, width = Size(video, frame-> channel, 0),
       int
height = Size(video, frame-> channel, 1);
       struct passwd *pswd;
       long
             clock;
       Dprintf("PostScript\n");
       sprintf(file_name, "%s%s/%s.ps\0", global->home, POST_DIR, video->name);
       fp = fopen(file_name, "w");
       fprintf(fp, "% %!PS-Adobe-1.0\n");
       pswd = getpwuid (getuid ());
       (void) XmuGetHostname (hostname, sizeof hostname);
       fprintf(fp, "%%% %Creator: %s: %s (%s)\n", hostname,pswd->pw name,
pswd->pw gecos);
       fprintf(fp, "%%%%Title: %s\n", video-> name);
```

```
fprintf(fp, "% % % BoundingBox: 0 0 %d %d\n", width, height);
                      fprintf(fp, "%%% CreationDate: %s",(time (&clock), ctime (&clock)));
                      fprintf(fp, "%%%%EndComments\n");
                     fprintf(fp, "%d %d scale\n", width, height);
                     fprintf(fp, "%d %d 8 image_print\n", width, height);
                     GetFrame(video, frame-> frame);
                     for(y=0;y < height;y++) {
                                          for(x=0;x < width;x++) {
                                                                                      X, Y, oct, data;
                                                                int
                                                                if (video-> trans.type = = TRANS Wave) {
CvtIndex(x,y,width,height,video-> trans.wavelet.space[0],&X,&Y,&oct);
data = 128 + Round(video-> data[frame-> channel %3][frame-> frame][Y*video-> size[0] +
X]*(oct = = video- > trans. wavelet. space[0]?1:4), video- > precision);
                                                           } else
data = 128 + Round(video-> data[frame-> channel %3][frame-> frame][y*video-> size[0] + (video-> channel %3)[frame-> frame][y*video-> channel %3)[frame-> frame][y*video-> frame][y*video-> frame][y*video-> frame][y*video-> frame][y*video-> frame][y*video
x], video-> precision);
                                                                fprintf(fp, \%02x\%, data < 0.0: data > 255.25: data);
                                          fprintf(fp, "\n");
                      }
                      FreeFrame(video, frame-> frame);
                      fclose(fp);
}
void
                     Spectrum(w,closure,call_data)
 Widget
                                           w:
 caddr_t
                                           closure, call_data;
```

```
{
      Frame frame = (Frame)closure;
                    *dpy = XtDisplay(global -> toplevel);
      Display
      XColor
                    xcolor[2], falsecolor;
      int
             i;
      Colormap
cmap=ChannelCmap(frame->channel,frame->video->type,frame->video->gamma);
      Dprintf("Spectrum\n");
      falsecolor.flags = DoRed | DoGreen | DoBlue;
      XSynchronize(dpy, True);
      for(i=0;i<2+global->levels;i++)
             if (i>1) XStoreColor(dpy,cmap,&xcolor[i&1]); /* Restore old color */
             if (i < global-> levels) {
                    xcolor(i&1).pixel=i;
                    XQueryColor(dpy,cmap,&xcolor[i&1]);
                    falsecolor.pixel=i;
                    falsecolor.red = xcolor[i&1].red + 32512;
                    falsecolor.green=xcolor[i&1].green+32512;
                    falsecolor.blue=xcolor[i&1].blue+32512;
                    XStoreColor(dpy,cmap,&falsecolor);
      XSynchronize(dpy,False);
}
```

```
source/icon3.c
```

```
/*
      Create Icons/Menus and set Callbacks
*/
              "../include/xwave.h"
#include
/*
      Function Name:
                            FindIcon
      Description: Finds IconRec entry from name in global icon array
       Arguments:
                    icon_name - name of icon bitmap
       Returns:
                    pointer to IconRec with the same name as icon_name
 */
      FindIcon(icon_name)
Icon
String icon_name;
{
       int
              i;
              icon=NULL;
       Icon
       for (i=0;i < global > no_icons;i++)
              if (!strcmp(global->icons[i].name,icon_name)) icon=&global->icons[i];
       return(icon);
}
       FillForm(parent, number, items, widgets, callbacks)
void
int
       number;
```

```
items[];
 FormItem
 Widget
                                                                                                                                           parent, widgets[];
 XtCallbackRec
                                                                                                                                                                                                                 callbacks[];
{
                                                                                                                                             args[10];
                                                                        Arg
                                                                                                                                             i, call i=0;
                                                                       int
                                                                        for(i=0;i < number;i++)
                                                                                                                                                                                                                   argc=0, *view=(int *)items[i].hook;
                                                                                                                                              int
                                                                                                                                              char
                                                                                                                                                                                                                   text(STRLEN];
                                                                                                                                              float top;
                                                                                                                                              NumInput
                                                                                                                                                                                                                                                                                          num = (NumInput)items[i].hook;
                                                                                                                                              FloatInput
                                                                                                                                                                                                                                                                                          fit = (FloatInput)items[i].hook;
                                                                                                                                              Message
                                                                                                                                                                                                                                                                                          msg = (Message)items[i].hook;
                                                                                                                                                WidgetClass
   class [15] = \{label Widget Class, command Widget Class, command Widget Class, ascii Text Widget Class, asc
    tClass,
    menuButton Widget Class, menuButton Widget Class, viewport Widget Class, toggle Widget Clas
    command Widget Class, command Widget Class, command Widget Class, label Widget Class, and the command Widget Class, command Widget
                                                                                                                                                                                                                    scrollbarWidgetClass, labelWidgetClass, formWidgetClass};
                                                                                                                                                Boolean
    call[15] = \{False, True, True, False, False, False, False, True, True, True, True, False, F
      e,False};
                                                                                                                                                if (items[i].fromHoriz!=0) {
                                                                                                                                                                                                                    XtSetArg(args[argc], XtNfromHoriz, widgets[items[i].fromHoriz-1]);
      argc++;
                                                                                                                                                 }
```

```
if (items[i].fromVert!=0) {
                     XtSetArg(args[argc], XtNfromVert, widgets[items[i].fromVert-1]);
argc++;
              }
              switch(items[i].type) { /* Initialise contents */
              case FW yn:
                     items[i].contents = *(Boolean *)items[i].hook?"confirm":"cancel";
              case FW up:
                     items[i].contents = "up";
                     break;
              case FW down:
                     items[i].contents = "down";
                     break;
              case FW integer:
                     sprintf(text, num-> format, *num-> value);
                     items[i].contents=text;
                     break:
              case FW float:
                     sprintf(text, flt-> format, *flt-> value);
                     items[i].contents = text;
                     break;
              }
              switch(items[i].type) { /* Set contents */
              case FW_label: case FW_command: case FW_button: case FW integer:
case FW_float:
                     XtSetArg(args[argc], XtNlabel, items[i].contents); argc + +;
                     break:
              case FW down: case FW up: case FW yn: case FW toggle: case
FW icon: case FW icon button: {
                            icon = FindIcon(items[i].contents);
```

```
if (icon = = NULL) {
                            XtSetArg(args[argc], XtNlabel, items[i].contents); argc++;
                     } else {
                            XtSetArg(args[argc],XtNbitmap,icon->pixmap); argc++;
                            XtSetArg(args[argc], XtNheight, icon->height+2); argc++;
                            XtSetArg(args[argc], XtN width, icon-> width +2); argc + +;
                     } break;
              }
              switch(items[i].type) { /* Individual set-ups */
             case FW text:
                     XtSetArg(args[argc], XtNstring, msg-> info.ptr); argc++;
                     XtSetArg(args[argc], XtNeditType, msg->edit); argc++;
                     XtSetArg(args[argc], XtNuseStringInPlace, True); argc + +;
                     XtSetArg(args[argc], XtNlength, msg-> size); argc++;
                     break:
              case FW button: case FW icon button:
                     XtSetArg(args[argc], XtNmenuName, (String) items[i].hook);
argc++;
                     break:
              case FW_toggle:
                     if ((int)items[i].hook = = 0) {
                            XtSetArg(args[argc],XtNradioData,1); argc++;
                     } else {
                            caddr t radioData;
                            Arg
                                   radioargs[1];
                                         radioGroup = widgets[(int)items[i].hook-1];
                            Widget
                            XtSetArg(radioargs[0], XtNradioData, &radioData);
                            XtGetValues(radioGroup, radioargs, ONE);
XtSetArg(args[argc],XtNradioData,(caddr t)((int)radioData+1)); argc++;
```

```
XtSetArg(args[argc], XtNradioGroup, radioGroup); argc++;
                     }
                     break:
             case FW_scroll:
                     top = (float)(flt-> value-flt-> min)/(flt-> max-flt-> min);
                     XtSetArg(args[argc], XtNtopOfThumb, &top); argc + +;
                     XtSetArg(args[argc], XtNjumpProc,&callbacks[call_i]); argc++;
                     while(callbacks[call_i].callback! = NULL) call_i++;
                            call_i++;
                     break;
             case FW view:
                     if (view! = NULL) {
                            XtSetArg(args[argc], XtNwidth, view[0]); argc++;
                            XtSetArg(args[argc], XtNheight, view[1]); argc + +;
                     }
                     break;
             }
widgets[i] = XtCreateManagedWidget(items[i].name,class[(int)items[i].type],parent,args,ar
gc);
              switch(items[i].type) { /* Post processing */
             case FW_toggle:
                     if (items[i].hook = = NULL) { /* Avoids Xaw bug */
                            XtSetArg(args[0], XtNradioGroup, widgets[i]);
                            XtSetValues(widgets[i], args, ONE);
                     break;
              case FW_text: {
                     XFontStruct *font;
                            text_args[1];
                     Arg
                     msg-> widget = widgets[i];
```

```
XawTextDisplayCaret(msg-> widget, msg-> edit! = XawtextRead);
                     XtSetArg(text_args[0], XtNfont, & font);
                     XtGetValues(widgets[i],text args,ONE);
                     argc = 0;
                     if (msg->edit = = XawtextRead && msg->info.ptr[0]! = '\0')
XtSetArg(args[argc], XtNwidth, 4 + TextWidth(0, msg-> info.ptr, font));
                     else
XtSetArg(args[argc], XtNwidth, 4+msg->cols+(font->max_bounds.width+font->min_bo
unds.width)/2);
                     argc++;
XtSetArg(args[argc], XtNheight, 1 + msg-> rows*(font-> max_bounds.ascent + font-> max_
bounds.descent)); argc++;
                     XtSetValues(widgets[i],args,argc);
                     } break;
              case FW_button:
XtOverrideTranslations(widgets[i], XtParseTranslationTable(" < BtnDown > : reset()
NameButton() PopupMenu()"));
                     break:
              case FW down:
                     if (*num-> value = = num-> min) XtSetSensitive(widgets[i], False);
                     num-> widgets[0] = widgets[i];
                     break:
              case FW up:
                     if (*num-> value = = num-> max) XtSetSensitive(widgets[i], False);
                     num-> widgets[1] = widgets[i];
                     break:
              case FW_integer:
                     num-> widgets[2] = widgets[i];
                     break;
              case FW_scroll:
```

```
flt-> widgets[1] = widgets[i];
                                                                                 XawScrollbarSetThumb(widgets[i],top,0.05);
                                                                                 break:
                                                     case FW_float:
                                                                                 flt-> widgets[0] = widgets[i];
                                                                                 break;
                                                      } .
                                                      if (call[(int)items[i].type]) { /* Add Callbacks */
                                                                                 if (callbacks[call_i].callback!=NULL)
                                                                                                             XtAddCallbacks(widgets[i],XtNcallback,&callbacks[call i]);
                                                                                  while(callbacks[call_i].callback!=NULL) call i++;
                                                                                 call_i++;
                                                      }
                          }
}
                                                       ShellWidget(name,parent,type,cmap,callbacks)
Widget
 String name;
 Widget
                                                       parent;
 ShellWidgetType
                                                                                  type;
 Colormap
                                                       cmap;
 XtCallbackRec
                                                                                  callbacks[];
 {
                            Widget
                                                                                  shell;
                             Arg
                                                       args[3];
                             Position
                                                                                   x, y;
                                                                                  height = -2;
                             Dimension
                                                        argc = 0;
                             int
                             WidgetClass
  class[] = \{transientShellWidgetClass, transientShellWidgetClass, topLevelShellWidgetClass, part of the property of the prope
```

```
ullRightMenuWidgetClass};
      if (type = SW below | | type = SW over) {
             XtTranslateCoords(parent,0,0,&x,&y);
             if (type = SW below) {
                    XtSetArg(args[0],XtNheight,&height);
                    XtGetValues(parent, args, ONE);
             XtSetArg(args[argc], XtNx,x); argc++;
             XtSetArg(args[argc], XtNy, y + height + 2); argc + +;
       if (cmap!=NULL) {
             XtSetArg(args[argc], XtNcolormap, cmap); argc + +;
       }
       shell = XtCreatePopupShell(name,class[type],parent,args,argc);
       if (callbacks! = NULL) XtAddCallbacks(shell, XtNdestroyCallback, callbacks);
       return(shell);
}
Widget
             FormatWidget(name, parent)
String name;
Widget
             parent:
{
       return(XtCreateManagedWidget(name,formWidgetClass,parent,NULL,ZERO));
}
       FillMenu(parent, number, items, widgets, callbacks)
void
int ·
      number:
Menultem
             items[];
```

```
parent, widgets[];
Widget
XtCallbackRec
                     callbacks[];
{
              args[4];
       Arg
       int
              i, call i=0;
             icon = FindIcon("right");
       Icon
       for(i=0;i < number;i++) {
              int
                     argc = 0;
              XtSetArg(args[argc], XtNlabel, items[i].label); argc++;
              if (items[i].widgetClass = = smeBSBprObjectClass) {
                     XtSetArg(args[argc], XtNmenuName, items[i].hook); argc++;
                     XtSetArg(args[argc], XtNrightMargin, 4+icon-> width); argc++;
                     XtSetArg(args[argc], XtNrightBitmap, icon->pixmap); argc++;
              }
widgets[i] = XtCreateManagedWidget(items[i].name,items[i].widgetClass,parent,args,argc)
              if (items[i].widgetClass = = smeBSBObjectClass) { /* Add Callbacks */
                     XtAddCallbacks(widgets[i], XtNcallback, &callbacks[call i]);
                     while(callbacks[call_i].callback! = NULL) call_i++;
                     call i++;
              }
       }
}
       SimpleMenu(w,closure,call_data)
void
Widget
              closure, call_data;
caddr t
```

```
{
       int
              *hook=(int *)closure, no child, child, argc=0;
       Widget
                     menu = XtParent(w), button;
       WidgetList
                     children:
             *label;
       char
       Arg
              args[3];
       XtSetArg(args[argc],XtNlabel,&label); argc++;
       XtGetValues(w,args,argc); argc=0;
       XtSetArg(args[argc], XtNchildren, &children); argc + +;
       XtSetArg(args[argc], XtNnumChildren, &no child); argc++;
       XtSetArg(args[argc], XtNbutton, &button); argc + +;
       XtGetValues(menu, args, argc); argc=0;
       for(child=0;children[child]!=w && child < no_child;) child++;
       if (w!=children[child]) Eprintf("SimpleMenu: menu error\n");
       *hook=child;
       XtSetArg(args[argc], XtNlabel, label); argc++;
       XtSetValues(button, args, argc);
}
       NumIncDec(w,closure,call_data)
void
Widget
              w;
              closure, call data;
caddr t
{
       Numinput
                    data = (NumInput) closure;
       Arg
              args[1];
              text[STRLEN];
       char
       *data > value + = (w = = data > widgets[0])?-1:1;
       sprintf(text,data-> format, *data-> value);
```

```
if (data->min = = *data-> value) XtSetSensitive(data-> widgets[0], False);
       else XtSetSensitive(data-> widgets[0], True);
       if (data -> max == *data -> value) XtSetSensitive(data -> widgets[1], False);
       else XtSetSensitive(data-> widgets[1],True);
       XtSetArg(args[0], XtNlabel, text);
       XtSetValues(data-> widgets[2], args, ONE);
}
       FloatIncDec(w, closure, call data)
Widget
              w;
caddr_t
              closure, call_data;
{
       FloatInput
                     data = (FloatInput)closure;
       Arg
              args[1];
       char
              text[STRLEN];
       float
            percent = *(float *)call_data;
       *data-> value = data-> min + (double) percent * (data-> max-data-> min);
       sprintf(text,data-> format, *data-> value);
       XtSetArg(args[0], XtNlabel, text);
      XtSetValues(data-> widgets[0], args, ONE);
}
/*
       Function Name:
                            ChangeYN
       Description: Toggle YN widget state
       Arguments:
                    w - toggling widget
                            closure - pointer to boolean state
                            call data - not used
       Returns:
                     none.
*/
```

```
void
       ChangeYN(w,closure,call_data)
Widget
caddr_t
              closure, call_data;
{
                      *bool = (Boolean *)closure;
       Boolean
              icon=FindIcon((*bool != True)?"confirm":"cancel");
       Icon
              args[4];
       Arg
              argc = 0;
       int
       *bool = ! *bool;
       XtSetArg(args[argc], XtNbitmap, icon->pixmap); argc++;
       XtSetArg(args[argc], XtNheight, icon-> height +2); argc + +;
       XtSetArg(args[argc], XtN width, icon-> width +2); argc + +;
       XtSetValues(w,args,argc);
.}
int
       TextWidth(max,text,font)
int
       max;
String text;
XFontStruct *font;
{
       int
              i = 0, j
       while(text[i]!='\0') {
                     width:
              int
              for(j=0;text[i+j]!='\0' \&\& text[i+j]!='\n';) j++;
              width=XTextWidth(font,&text[i],j);
```

max = max > width?max:width;

Copyright 1987, 1988 by Digital Equipment Corporation, Maynard, Massachusetts, and the Massachusetts Institute of Technology, Cambridge, Massachusetts.

All Rights Reserved

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the names of Digital or MIT not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

DIGITAL DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING

ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL

DIGITAL BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR

ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS.

WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION,

ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS

SOFTWARE.

```
/*
 * Image.c - Image widget
 */
#define XtStrlen(s)
                            ((s) ? strlen(s) : 0)
#include < stdio.h>
#include < ctype.h >
#include < X11/IntrinsicP.h>
#include < X11/StringDefs.h>
#include <X11/Xaw/XawInit.h>
#include "../include/ImageP.h"
#define streq(a,b) (strcmp( (a), (b) ) = = 0)
 * Full class record constant
/* Private Data */
static char defaultTranslations[] =
       " < Btn1Down >: notify()\n\
        < Btn1Motion > : notify()\n\
        <Btn1Up>: notify()*;
#define offset(field) XtOffset(ImageWidget, field)
static XtResource resources[] = {
       {XtNbitmap, XtCPixmap, XtRBitmap, sizeof(Pixmap),
```

```
offset(image.pixmap), XtRImmediate, (caddr t)None},
       {XtNcallback, XtCCallback, XtRCallback, sizeof(XtPointer),
       offset(image.callbacks), XtRCallback, (XtPointer)NULL},
};
static void Initialize();
static void Resize();
static void Redisplay();
static Boolean SetValues();
static void ClassInitialize();
static void Destroy();
static XtGeometryResult QueryGeometry();
             Notify(), GetBitmapInfo();
static void
static XtActionsRec
                             actionsList[] = {
       {"notify",
                     Notify}.
};
ImageClassRec imageClassRec = {
/* core class fields */
#define superclass
                             (&simpleClassRec)
   /* superclass
                                    (WidgetClass) superclass,
   /* class name
                             */
                                     "Image",
   /* widget_size
                             */
                                    sizeof(ImageRec),
                             */
                                    ClassInitialize.
   /* class initialize
   /* class_part_initialize
                             */
                                    NULL,
   /* class_inited
                             */
                                    FALSE,
   /* initialize
                             */
                                    Initialize,
   /* initialize hook
                             */
                                    NULL,
                             */
                                    XtInheritRealize,
   /* realize
```

```
*/
                                  actionsList.
  /* actions
  /* num_actions
                           */
                                  XtNumber(actionsList),
                           */
                                  resources,
  /* resources
                           */
                                  XtNumber(resources),
  /* num resources
                                  NULLQUARK,
                           */
  /* xrm class
                                  */
                                         TRUE,
  /* compress motion
                           */
                                  TRUE.
  /* compress exposure
                                  TRUE,
                           */
  /* compress_enterleave
                                  */
                                         FALSE,
  /* visible interest
                           */
  /* destroy
                                  Destroy,
                           */
                                  Resize.
  /* resize
                                  Redisplay,
                            */
  /* expose
  /* set_values
                            */.
                                  SetValues,
                                         NULL.
  /* set values hook
                            */
                                  XtInheritSetValuesAlmost,
  /* set values_almost
                                  */
                                         NULL,
  /* get_values_hook
                                  NULL.
                            */
  /* accept focus
                                  XtVersion,
                            */
  /* version
                                  NULL.
                            */
  /* callback_private
                                  */
  /* tm table
                                         defaultTranslations.
                                  */
                                         QueryGeometry,
  /* query_geometry
                                  XtInheritDisplayAccelerator,
   /* display accelerator
                            */
                            */
                                  NULL
   /* extension
 },
/* Simple class fields initialization */
 {
                                         XtInheritChangeSensitive
                                  */
   /* change_sensitive
 }
};
WidgetClass imageWidgetClass = (WidgetClass)&imageClassRec;
```

```
* Private Procedures
static void ClassInitialize()
      extern void XmuCvtStringToBitmap();
   static XtConvertArgRec screenConvertArg[] = {
      {XtWidgetBaseOffset, (caddr_t) XtOffset(Widget, core.screen),
          sizeof(Screen *)}
   };
   XawInitializeWidgetSet();
      XtAddConverter("String", "Bitmap", XmuCvtStringToBitmap,
             screenConvertArg, XtNumber(screenConvertArg));
} /* ClassInitialize */
/* ARGSUSED */
static void Initialize(request, new)
Widget request, new;
   ImageWidget iw = (ImageWidget) new;
      Dprintf("ImageInitialize\n");
      if (iw-> image.pixmap = = NULL)
             XtErrorMsg("NoBitmap", "asciiSourceCreate", "XawError",
             "Image widget has no bitmap.", NULL, 0);
      GetBitmapInfo(new);
      if (iw->image.map_width<=0 || iw->image.map_height<=0)
             XtErrorMsg("NoDimension", "asciiSourceCreate", "XawError",
             "Image widget illegal map dimension.", NULL, 0);
```

```
if (iw->core.width == 0) iw->core.width=iw->image.map width;
      if (iw->core.height == 0) iw->core.height=iw->image.map height;
   (*XtClass(new)->core class.resize) ((Widget)iw);
} /* Initialize */
/*
* Repaint the widget window
*/
/* ARGSUSED */
static void Redisplay(w, event, region)
   Widget w;
   XEvent *event;
   Region region;
{
  ImageWidget iw = (ImageWidget) w;
      Dprintf("ImageRedisplay\n");
      if (region != NULL &&
      XRectInRegion(region, 0, 0,
                 iw->image.map_width, iw->image.map_height)
          == RectangleOut)
     return;
      XCopyArea(
              XtDisplay(w), iw-> image.pixmap, XtWindow(w),
DefaultGC(XtDisplay(w), XDefaultScreen(XtDisplay(w))),
             0, 0, iw->image.map width, iw->image.map height, 0, 0);
```

```
static void Resize(w)
   Widget w;
{
   ImageWidget iw = (ImageWidget)w;
      Dprintf("ImageResize\n");
}
/*
 * Set specified arguments into widget
 */
static Boolean SetValues(current, request, new, args, num_args)
   Widget current, request, new;
   ArgList args;
   Cardinal *num args;
{
   ImageWidget curiw = (ImageWidget) current;
   ImageWidget reqiw = (ImageWidget) request;
   ImageWidget newiw = (ImageWidget) new;
   Boolean redisplay = False;
   /* recalculate the window size if something has changed. */
      if (curiw-> image.pixmap! = newiw-> image.pixmap)
XFreePixmap(XtDisplay(curiw),curiw->image.pixmap);
      GetBitmapInfo(newiw);
      newiw->core.width=newiw->image.map width;
      newiw->core.height=newiw->image.map height;
      redisplay = True;
   return redisplay | | XtlsSensitive(current) != XtlsSensitive(new);
}
```

WO 94/23385

```
static void Destroy(w)
   Widget w;
   ImageWidget lw = (ImageWidget)w;
      Dprintf("ImageDestroy\n");
}
static XtGeometryResult QueryGeometry(w, intended, preferred)
   Widget w;
   XtWidgetGeometry *intended, *preferred;
{
   register ImageWidget iw = (ImageWidget)w;
   preferred->request_mode = CWWidth | CWHeight;
   preferred-> width = iw-> image.map_width;
   preferred->height = iw->image.map_height;
   if ( ((intended-> request_mode & (CWWidth | CWHeight))
             == (CWWidth | CWHeight)) &&
        intended-> width == preferred-> width &&
        intended->height == preferred->height)
      return XtGeometry Yes;
   else if (preferred-> width == w-> core.width &&
          preferred->height == w->core.height)
      return XtGeometryNo;
   else
      return XtGeometryAlmost;
}
static void GetBitmapInfo(w)
```

```
Widget
             w;
{
      ImageWidget iw=(ImageWidget)w;
      unsigned int depth, bw;
      Window
                   root;
      int
             x, y;
      unsigned int width, height;
           buf[BUFSIZ];
      char
      if (iw->image.pixmap!= None) {
             if
(!XGetGeometry(XtDisplayOfObject(w),iw->image.pixmap,&root,&x,&y,&width,&heig
ht,&bw,&depth)) {
                   sprintf(buf, "ImageWidget: %s %s \"%s\".", "Could not",
                    "get Bitmap geometry information for Image ",
                   XtName(w));
                   XtAppError(XtWidgetToApplicationContext(w), buf);
             }
             iw-> image.map_width=(Dimension)width;
             iw->image.map height=(Dimension)height;
      }
}
/*
       Action Procedures
 */
             Notify(w,event,params,num_params)
static void
Widget
             w;
XEvent
             *event;
```

```
String *params;
Cardinal
              *mum_params;
{
       ImageWidget iw=(ImageWidget)w;
       XButtonEvent
                            *buttonevent = &event- > xbutton;
              posn[2] = \{buttonevent-> x, buttonevent-> y\};
       int
       if (iw->image.map\_width < =posn[0] \mid \mid posn[0] < 0 \mid \mid
              iw->image.map_height<=posn[1] || posn[1]<0) Dprintf("No
ImageNotify\n");
       else {
              Dprintf("ImageNotify\n");
              XtCallCallbackList(w,iw->image.callbacks,posn);
       }
}
```

source/ImpKlicsTestSA.c

```
/*
      Test harness for KlicsFrameSA() in Klics.SA
*/
#include
             "xwave.h"
#include
             "KlicsSA.h"
void
      ImpKlicsTestSA(w,closure,call_data)
Widget
             w:
caddr t
             closure, call_data;
{
      int
             sizeY=SA WIDTH*SA HEIGHT,
                   sizeUV=SA_WIDTH*SA_HEIGHT/4;
      short *dst[3] = {
             (short *)MALLOC(sizeof(short)*sizeY),
             (short *)MALLOC(sizeof(short)*sizeUV),
             (short *)MALLOC(sizeof(short)*sizeUV),
      }, *src[3];
      Video video=(Video)MALLOC(sizeof(VideoRec));
      int
             i, z;
      char
             file_name[STRLEN];
      Bits
             bfp;
      Boolean
                   stillvid;
      strcpy(video-> name,((XawListReturnStruct *)call_data)-> string);
```

WO 94/23385

```
sprintf(file name, "%s%s/%s%s\0", global->home, KLICS_SA_DIR, video->name, KLICS
SA EXT);
      bfp=bopen(file_name, "r"); '
      bread(&stillvid, 1, bfp);
      bread(&video->size[2],sizeof(int)*8,bfp);
      video->data[0] = (short **)MALLOC(sizeof(short *)*video->size[2]);
      video->data[1]=(short **)MALLOC(sizeof(short *)*video->size[2]);
      video->data[2]=(short **)MALLOC(sizeof(short *)*video->size[2]);
       video->disk=False;
       video > type = YUV;
       video - > size[0] = SA_WIDTH;
      video->size[1]=SA HEIGHT;
       video > UVsample[0] = 1;
       video > UVsample[1] = 1;
       video->trans.type=TRANS_None;
       for(z=0;z < video-> size[2];z++) {
             NewFrame(video,z);
              src[0] = video - > data[0][z];
              src[1] = video -> data[1][z];
              src[2] = video -> data[2][z];
             KlicsFrameSA(z = 0 | stillvid?STILL:SEND,src,dst,bfp);
              SaveFrame(video,z);
              FreeFrame(video,z);
       bclose(bfp);
       video-> next = global-> videos;
       global-> videos = video;
       XtFree(dst[0]);
       XtFree(dst[1]);
       XtFree(dst[2]);
}
```

source/ImportKlics.c

```
Importing raw Klics binary files
#include
              "xwave.h"
#include
              "Klics.h"
              bopen();
extern Bits
extern void bclose(), bread(), bwrite(), bflush();
              SkipFrame();
extern void
              HuffRead();
extern int
extern Boolean
                     BlockZero();
extern void
              ZeroCoeffs();
              ReadInt();
extern int
extern int
              Decide();
                     DecideDouble();
extern double
              BoolToken(bfp)
Boolean
Bits
       bfp;
{
       Boolean
                     token;
       bread(&token,1,bfp);
       return(token);
}
```

```
void HuffBlock(block,bfp)
Block block;
Bits
      bfp;
{
             X, Y;
      int
      for(X=0;X < BLOCK;X++) for(Y=0;Y < BLOCK;Y++)
             block[X][Y] = HuffRead(bfp);
}
      PrevBlock(old,addr,x,y,z,oct,sub,channel,ctrl)
Block old, addr;
int
      x, y, z, oct, sub, channel;
CompCtrl
             ctrl;
{
             X. Y:
       int
      for(X=0;X<BLOCK;X++) for(Y=0;Y<BLOCK;Y++) {
addr[X][Y] = Access((x < < 1) + X, (y < < 1) + Y, oct, sub, Size(ctrl->dst, channel, 0));
             old[X][Y] = ctrl-> dst-> data[channel][z][addr[X][Y]];
       }
}
      DeltaBlock(new,old,delta,step)
void
Block new, old, delta;
int
       step;
```

```
{
             X, Y;
       int
      for(X=0;X < BLOCK;X++) for(Y=0;Y < BLOCK;Y++)
new[X][Y] = old[X][Y] + delta[X][Y] * step + (delta[X][Y]! = 0?negif(delta[X][Y] < 0, (step-1))
> > 1):0);
void UpdateBlock(new,addr,z,channel,ctrl)
int
      z, channel;
Block new, addr;
CompCtrl
             ctrl;
{
             X, Y;
      int
      for(X=0;X < BLOCK;X++) for(Y=0;Y < BLOCK;Y++)
             ctrl-> dst-> data[channel][z][addr[X][Y]] = (short)new[X][Y];
}
void ReadKlicsHeader(ctrl)
CompCtrl
             ctri;
{
      KlicsHeaderRec
                           head:
      int
             i;
       Video dst=ctrl->dst;
       fread(&head, sizeof(KlicsHeaderRec), 1, ctrl-> bfp-> fp);
```

```
ctrl-> stillvid = head.stillvid:
       ctrl->auto q=head.auto q;
       ctrl->buf switch=head.buf switch;
      ctrl->quant_const=head.quant const;
      ctrl->thresh const=head.thresh const;
      ctrl->cmp const=head.cmp const;
      ctrl - ps = head.fps;
      for(i=0;i<5;i++) ctrl-> base_factors[i]=head.base_factors[i];
      ctrl->diag factor=head.diag factor;
      ctrl->chrome factor=head.chrome factor;
       ctrl-> decide = head.decide;
       strcpy(dst->name,ctrl->bin name);
      dst->type=head.type;
       dst - > disk = head.disk:
      dst-> gamma = head.gamma;
      dst-> rate = head.rate;
      dst-> start = head.start;
       for(i=0;i<3;i++) dst-> size[i]=head.size[i];
      for(i=0;i<2;i++) dst-> UVsample[i] = head.UVsample[i];
      dst-> trans = head. trans:
      dst->precision=head.precision;
      for(i=0; i < (dst-> type = = MONO?1:3); i++)
             dst->data[i]=(short **)MALLOC(dst->size[2]*sizeof(short *));
}
      WriteKlicsHeader(ctrl)
void
CompCtrl
             ctrl;
{
     · KlicsHeaderRec
                            head:
      int
             i:
```

```
head.stillvid = ctrl-> stillvid;
       head.auto_q = ctrl > auto_q;
       head.buf_switch=ctrl->buf switch;
       head.quant_const = ctrl-> quant_const;
       head.thresh_const = ctrl-> thresh_const;
       head.cmp_const=ctrl->cmp_const;
       head. fps = ctrl - > fps;
       for(i=0; i<5; i++) head.base_factors[i] = ctrl-> base_factors[i];
       head.diag_factor=ctrl->diag_factor;
       head.chrome_factor=ctrl->chrome_factor;
       head.decide = ctrl-> decide;
       head.type=ctrl->dst->type;
       head.disk=ctrl->dst->disk;
       head.gamma = ctrl->dst->gamma;
       head.rate = ctrl-> dst-> rate:
       head.start=ctrl->dst->start;
       for(i=0;i<3;i++) head.size[i] = ctrl->dst->size[i];
       for(i=0;i<2;i++) head. UVsample[i] = ctrl->dst-> UVsample[i]:
       head.trans=ctrl->dst->trans;
       head.precision = ctrl-> dst-> precision;
       fwrite(&head, sizeof(KlicsHeaderRec), 1, ctrl-> bfp-> fp);
}
void
       KlicsTree(mode,x,y,z,oct,sub,channel,ctrl)
int
       mode, x, y, z, oct, sub, channel;
CompCtrl
              ctrl;
{
       Block addr, old, new, delta, zero_block=\{\{0,0\},\{0,0\}\}\;
       double
                     norms[3] = {ctrl-> quant_const,ctrl-> thresh_const,ctrl-> cmp_const};
       int
              step;
```

```
PrevBlock(old,addr,x,y,z,oct,sub,channel,ctrl);
      if (mode! = VOID) {
             CalcNormals(ctrl,oct,sub,channel,norms);
             step = norms[0] < 1.0?1:(int)norms[0];
             if (mode = STILL | BlockZero(old)) {
                    if (BoolToken(ctrl->bfp)) { /* NON_ZERO_STILL */
                          Dprintf("NON_ZERO_STILL\n");
                          HuffBlock(delta,ctrl->bfp);
                          DeltaBlock(new,old,delta,step);
                          UpdateBlock(new,addr,z,channel,ctrl);
                    } else {
                          Dprintf("ZERO_STILL\n");
                          mode = STOP:
                                                            /* ZERO_STILL */
                   }
             } else {
                   if (!BoolToken(ctrl->bfp)) {
                                                     /* BLOCK SAME */
                          Dprintf("BLOCK SAME\n");
                          mode=STOP;
                   } else {
                          if (!BoolToken(ctrl->bfp)) {
                                                            /* ZERO_VID */
                                 Dprintf("ZERO_VID\n");
                                 ZeroCoeffs(ctrl->dst->data[channel][z],addr);
                                 mode = VOID:
                          } else {
                                                                         /*
BLOCK CHANGE */
                                 Dprintf("BLOCK_CHANGE\n");
                                 HuffBlock(delta,ctrl->bfp);
                                 DeltaBlock(new,old,delta,step);
                                 UpdateBlock(new,addr,z,channel,ctrl);
                          }
                   }
            }
```

```
} else {
             if (BlockZero(old)) mode=STOP;
             else {
                    ZeroCoeffs(ctrl->dst->data[channel][z],addr);
                    mode = VOID;
             }
      }
      if (oct > 0 && mode! = STOP) {
                           decend = mode = = VOID?True:BoolToken(ctrl-> bfp);
             Boolean
                    X, Y;
             int
             Dprintf("x = %d, y = %d, oct = %d sub = %d mode
%d\n",x,y,oct,sub,mode);
             if (decend) {
                    if (mode! = VOID) Dprintf("OCT_NON ZERO\n");
                    for(Y=0;Y<2;Y++) for(X=0;X<2;X++)
                           KlicsTree(mode, x+2+X, y+2+Y, z, oct-1, sub, channel, ctrl);
             } else if (mode!=VOID) Dprintf("OCT ZERO\n");
      }
}
      KlicsLPF(mode,z,ctrl)
void
CompCtrl
             ctrl;
int
      mode, z;
{
      Block addr, old, new, delta;
      int
             channel, channels = ctrl > dst > type = = MONO?1:3, x, y,
                    octs_lum=ctrl->dst->trans.wavelet.space[0],
size[2] = {Size(ctrl-> dst,0,0) > octs_lum+1, Size(ctrl-> dst,0,1) > octs_lum+1};
```

```
for(y = 0; y < size[1]; y + +) for(x = 0; x < size[0]; x + +) {
              Boolean
                            lpf loc = True;
              if (mode! = STILL) {
                     lpf loc = BoolToken(ctrl- > bfp); /*
LPF LOC ZERO/LPF_LOC_NON_ZERO */
Dprintf("%s\n",lpf_loc?"LPF_LOC_NON_ZERO":"LPF_LOC_ZERO");
              if (lpf_loc) for(channel=0;channel<channels;channel++) {
                     int
octs = ctrl->dst-> trans. wavelet.space[ctrl->dst-> type = = YUV && channel! = 0?1:0],
                                   X, Y, step, value, bits = 0;
                     double
norms[3]={ctrl->quant_const,ctrl->thresh const,ctrl->cmp const};
                     PrevBlock(old,addr,x,y,z,octs-1,0,channel,ctrl);
                     CalcNormals(ctrl,octs-1,0,channel,norms);
                     step = norms[0] < 1.0?1:(int)norms[0];
                     if (mode = = STILL) {
                            for(bits=0.
value = ((1 < 8 + ctrl - > dst - > precision) - 1)/step; value! = 0; bits + +)
                                   value = value > > 1:
                            for(X=0;X < BLOCK;X++) for(Y=0;Y < BLOCK;Y++)
                                   delta[X][Y] = ReadInt(bits,ctrl->bfp);
                            DeltaBlock(new,old,delta,step);
                            UpdateBlock(new,addr,z,channel,ctrl);
                     } else {
                            if (BoolToken(ctrl->bfp)) { /*
LPF ZERO/LPF NON ZERO */
                                   Dprintf("LPF_NON_ZERO\n");
                                   HuffBlock(delta,ctrl->bfp);
```

```
DeltaBlock(new,old,delta,step);
                                   UpdateBlock(new,addr,z,channel,ctrl);
                            } else Dprintf("LPF_ZERO\n");
                     }
              }
       }
}
void
       KlicsFrame(ctrl,z)
CompCtri
              ctrl:
int
       z;
{
       Video dst=ctrl->dst;
       int
              sub, channel, x, y, mode=ctrl->stillvid ||z==0?STILL:SEND|
                    octs_lum = dst-> trans. wavelet.space[0],
size[2] = {Size(dst,0,0) > 1 + octs_lum, Size(dst,0,1) > 1 + octs_lum};
       NewFrame(dst,z);
      CopyFrame(dst,z-1,z,ctrl-> stillvid ||z==0\rangle;
       if (z!=0 \&\& ctrl-> auto q) {
ctrl->quant_const+=(double)(HISTO/2+ReadInt(HISTO_BITS,ctrl->bfp))*HISTO_DE
LTA*2.0/HISTO-HISTO_DELTA;
             ctrl->quant_const=ctrl->quant_const<0.0?0.0:ctrl->quant_const;
             Dprintf("New quant %f\n",ctrl->quant_const);
      KlicsLPF(mode,z,ctrl);
     for(y=0;y < size[1];y++) for(x=0;x < size[0];x++) {
             if (BoolToken(ctrl->bfp)) {
```

```
Dprintf("LOCAL NON ZERO\n");
                    for(channel = 0; channel < (dst-> type = = MONO?1:3); channel + +) {
                           int
                                  octs = dst-> trans. wavelet. space[dst-> type = = YUV
&& channel! = 0?1:0];
                           if (BoolToken(ctrl->bfp)) {
                                  Dprintf("CHANNEL_NON ZERO\n");
                                  for(sub=1;sub < 4;sub++)
                                         KlicsTree(mode, x, y, z, octs-1, sub, channel, ctrl);
                           } else Dprintf("CHANNEL_ZERO\n");
              } else Dprintf("LOCAL ZERO\n");
      }
}
      ImportKlics(w,closure,call_data)
void
Widget
              closure, call data;
caddr_t
              file name[STRLEN];
       char
       CompCtrlRec ctrl;
       int
              i, z;
       ctrl.dst=(Video)MALLOC(sizeof(VideoRec));
     strcpy(ctrl.bin_name,((XawListReturnStruct *)call_data)->string);
sprintf(file_name, "%s%s/%s%s\0",global->home,KLICS_DIR,ctrl.bin_name,KLICS_EX
T);
       ctrl.bfp=bopen(file name, "r");
       ReadKlicsHeader(&ctrl);
```

}

```
if (ctrl.dst->disk) SaveHeader(ctrl.dst);
for(z=0;z<ctrl.dst->size[2];z++) {
    if (z==0 || !ctrl.buf_switch) KlicsFrame(&ctrl.z);
    else {
        if (BoolToken(ctrl.bfp)) KlicsFrame(&ctrl.z);
        else SkipFrame(ctrl.dst,z);
    }
    if (z>0) {
        SaveFrame(ctrl.dst,z-1);
        FreeFrame(ctrl.dst,z-1);
    }
}
SaveFrame(ctrl.dst,ctrl.dst->size[2]-1);
FreeFrame(ctrl.dst,ctrl.dst->size[2]-1);
bclose(ctrl.bfp);
ctrl.dst->next=global->videos;
global->videos=ctrl.dst;
```

{

```
source/ImportKlicsSA.c
/*
      Importing raw Klics binary files
      Stand Alone version
#include
             "KlicsSA.h"
             Convolve();
extern void
/* useful X definitions */
typedef char Boolean;
#define True 1
#define False 0
#define String char*
             HuffReadSA();
extern int
extern Boolean
                   BlockZeroSA();
extern void ZeroCoeffsSA();
             ReadIntSA();
extern int
             DecideSA();
extern int
extern double
                   DecideDoubleSA();
             BoolTokenSA(bfp)
Boolean
Bits
      bfp;
```

```
Boolean
                    token;
      bread(&token,1,bfp);
      return(token);
}
void
      HuffBlockSA(block,bfp)
Block block;
Bits
      bfp;
{
             X, Y;
      int
      for(X=0;X < BLOCK;X++) for(Y=0;Y < BLOCK;Y++)
             block[X][Y] = HuffReadSA(bfp);
}
      PrevBlockSA(old,addr,x,y,oct,sub,channel,dst)
Block old, addr;
      x, y, oct, sub, channel;
short *dst[3];
{
            X, Y;
      int
      for(X=0;X<BLOCK;X++) for(Y=0;Y<BLOCK;Y++) {
            addr[X][Y] = AccessSA((x < < 1) + X,(y < < 1) + Y,oct,sub,channel);
            old[X][Y] = dst[channel][addr[X][Y]];
      }
}
```

```
void
      DeltaBlockSA(new,old,delta,step)
Block new, old, delta;
int
       step;
{
             X, Y;
      int
      for(X=0;X < BLOCK;X++) for(Y=0;Y < BLOCK;Y++)
new[X][Y] = old[X][Y] + delta[X][Y] * step + (delta[X][Y]! = 0?negif(delta[X][Y] < 0, (step-1))
> > 1):0);
}
void
      UpdateBlockSA(new,addr,channel,dst)
      channel;
int
Block new, addr;
short *dst[3];
{
      int
             X, Y;
      for(X=0;X < BLOCK;X++) for(Y=0;Y < BLOCK;Y++)
             dst[channel][addr[X][Y]] = (short)new[X][Y];
}
      KlicsTreeSA(mode,x,y,oct,sub,channel,dst,bfp,quant const)
void
      mode, x, y, oct, sub, channel;
int
      *dst[3];
short
Bits
      bfp;
```

```
double
             quant_const;
      Block addr, old, new, delta, zero_block=\{\{0,0\},\{0,0\}\}\;
      double
                    norms[3] = {quant const,thresh const,cmp const};
      int
             step;
      PrevBlockSA(old,addr,x,y,oct,sub,channel,dst);
      if (mode! = VOID) {
             CalcNormalsSA(oct, sub, channel, norms, quant_const);
             step = norms[0] < 1.0?1:(int)norms[0];
             if (mode = = STILL | | BlockZero(old)) {
                    if (BoolTokenSA(bfp)) { /* NON_ZERO_STILL */
                           Dprintf("NON_ZERO_STILL\n");
                           HuffBlockSA(delta,bfp);
                           DeltaBlockSA(new,old,delta,step);
                           UpdateBlockSA(new,addr,channel,dst);
                    } else { .
                           Dprintf("ZERO_STILL\n");
                           mode = STOP;
                                                             /* ZERO_STILL */
             } else {
                    if (!BoolTokenSA(bfp)) {
                                               /* BLOCK SAME */
                           Dprintf("BLOCK_SAME\n");
                           mode=STOP;
                    } else {
                           if (!BoolTokenSA(bfp)) {
                                                      /* ZERO VID */
                                  Dprintf("ZERO_VID\n");
                                  ZeroCoeffsSA(dst[channel],addr);
                                  mode = VOID;
                                                                          /*
                           } else {
BLOCK_CHANGE */
```

- 170 -

```
Dprintf("BLOCK CHANGE\n");
                                 HuffBlockSA(delta,bfp);
                                 DeltaBlockSA(new,old,delta,step);
                                 UpdateBlockSA(new,addr,channel,dst);
                          }
                   }
             }
      } else {
             if (BlockZeroSA(old)) mode=STOP;
            else {
                   ZeroCoeffsSA(dst[channel],addr);
                   mode = VOID;
             }
      }
      if (oct > 0 && mode! = STOP) {
             Boolean
                          decend = mode = = VOID?True:BoolTokenSA(bfp);
                   X, Y;
             int
             Dprintf(x = %d, y = %d, oct = %d sub = %d mode
%d\n",x,y,oct,sub,mode);
             if (decend) {
                   if (mode! = VOID) Dprintf("OCT_NON_ZERO\n");
                   for(Y=0;Y<2;Y++) for(X=0;X<2;X++)
KlicsTreeSA(mode, x*2+X, y*2+Y, oct-1, sub, channel, dst, bfp, quant_const);
            } else if (mode!=VOID) Dprintf("OCT_ZERO\n");
      }
}
void
      KlicsLPF_SA(mode,dst,bfp,quant_const)
int
      mode;
```

```
short *dst[3];
Bits
       bfp;
double
              quant_const;
{
       Block addr, old, new, delta;
              channel, channels = 3, x, y,
                     octs_lum = 3,
size[2] = \{SA\_WIDTH > octs\_lum + 1, SA\_HEIGHT > octs\_lum + 1\};
       for(y=0;y < size[1];y++) for(x=0;x < size[0];x++) {
              Boolean
                            lpf_loc = True;
              if (mode! = STILL) {
                    lpf_loc=BoolTokenSA(bfp); /*
LPF_LOC_ZERO/LPF_LOC_NON_ZERO */
Dprintf("%s\n",lpf_loc?"LPF_LOC_NON_ZERO":"LPF_LOC_ZERO");
              if (lpf_loc) for(channel = 0; channel < channels; channel + +) {
                            octs = channel! = 0?2:3,
                    int
                                  X, Y, step, value, bits=0;
                    double
                                  norms[3] = {quant_const, thresh_const, cmp_const};
                    PrevBlockSA(old,addr,x,y,octs-1,0,channel,dst);
                    CalcNormalsSA(octs-1,0,channel,norms,quant const);
                    step = norms[0] < 1.0?1:(int)norms[0];
                    if (mode = = STILL) {
                           for(bits=0.
value = ((1 < 8 + SA_PRECISION) - 1)/step; value! = 0; bits + +)
                                  value = value > > 1:
```

```
- 172 -
```

```
for(X=0;X < BLOCK;X++) for(Y=0;Y < BLOCK;Y++)
                                 delta[X][Y] = ReadIntSA(bits, bfp);
                          ·DeltaBlockSA(new,old,delta,step);
                           UpdateBlockSA(new,addr,channel,dst);
                    } else {
                          if (BoolTokenSA(bfp)) { /* LPF_ZERO/LPF_NON_ZERO
*/
                                 Dprintf("LPF_NON_ZERO\n");
                                 HuffBlockSA(delta,bfp);
                                 DeltaBlockSA(new,old,delta,step);
                                 UpdateBlockSA(new,addr,channel,dst);
                          } else Dprintf("LPF_ZERO\n");
                   · }
             }
      }
}
      KlicsFrameSA(mode,src,dst,bfp)
void
int
      mode:
short
      *src[3], *dst[3];
Bits
      bfp;
{
      int
             sub, channel, x, y, i,
                   octs_lum = 3,
size[2]={SA_WIDTH>>1+octs_lum,SA_HEIGHT>>1+octs_lum};
      double
                   quant_const;
      bread((char *)&quant_const.sizeof(double)*8,bfp);
      KlicsLPF_SA(mode,dst,bfp,quant const);
```

```
- 173 -
```

```
for(y = 0; y < size[1]; y + +) for(x = 0; x < size[0]; x + +) {
              if (BoolTokenSA(bfp)) {
                     Dprintf("LOCAL_NON_ZERO\n");
                     for(channel = 0; channel < 3; channel + +) {
                            int
                                   octs=channel!=0?2:3;
                            if (BoolTokenSA(bfp)) {
                                   Dprintf("CHANNEL_NON_ZERO\n");
                                   for(sub=1;sub<4;sub++)
KlicsTreeSA(mode,x,y,octs-1,sub,channel,dst,bfp,quant_const);
                            } else Dprintf("CHANNEL_ZERO\n");
              } else Dprintf("LOCAL_ZERO\n");
       }
       for(channel = 0; channel < 3; channel + +) {
              int
frame_size[2] = {SA\_WIDTH > (channel = = 0?0:1), SA\_HEIGHT > (channel = = 0?0:1)}
)},
                           frame_area = frame_size[0] * frame_size[1];
             for(i=0;i < frame_area;i++) src[channel][i]=dst[channel][i];
             Convolve(src[channel], False, frame_size, channel = = 0?3:2,0);
             for(i=0; i < frame_area; i++)
src[channel][i] = src[channel][i] > > SA_PRECISION;
      }
}
```

source/InitFrame.c

```
/*
      Initialise frame structure for Frame command widget
*/
#include
             "../include/xwave.h"
#define
             FRAME ICONS
                                 14
#define
             TRANS_MENU
#define
             COMP_MENU
                                 2
             CopyVideo();
extern void
extern void Compare();
             NAO;
extern void
extern void FrameDestroy();
extern void Examine();
extern void FramePointYN();
extern void
            FrameInfo();
            FrameMerge();
extern void
extern void
            Movie();
extern void
            PostScript();
            Select();
extern void
            Spectrum();
extern void
extern void NewPoint();
            Transform();
extern void
            Compress();
extern void
extern String *VideoCurrentList();
extern void
            KlicsSA();
      InitFrame
                   (w,closure,call_data)
void
```

```
Widget
             w;
caddr t
             closure, call data;
{
      XawListReturnStruct *name=(XawListReturnStruct *)call data;
      Video video = FindVideo(name-> string, global-> videos);
      Frame frame = (Frame)MALLOC(sizeof(FrameRec));
      Widget
                    shell[2], form, widgets[FRAME_ICONS],
trans_widgets[TRANS MENU], comp widgets[COMP MENU];
      Arg
             args[7];
      Pixmap
                   pixmap;
             view[2] = \{15 + video - > size[0], 15 + video - > size[1]\};
      FormItem
                   items[] = {
             {"frm_cancel",
                                 "frame_close",
                                                            0,0,FW icon, NULL),
             {"frm_copy", "copy",
                                                            1,0,FW_icon,NULL},
                                                            2,0,FW icon,NULL},
             {"frm_exam",
                                 "examine".
                                                     3,0,FW_icon,NULL},
             {"frm_point_yn", "point_y",
             {"frm_transform", "transform",
4,0,FW_icon_button, "frm_trans_menu"},
             {"frm_info_yn",
                                 "info".
5,0,FW icon,NULL},
             {"frm_merge",
                                 "merge",
                                                            6,0,FW_toggle,NULL},
             {"frm compress", "code",
7,0,FW icon button, "frm comp menu"},
             {"frm_movie",
                                 "movie",
                                                            8,0,FW_icon,NULL},
             {"frm_postscript", "postscript",
                                                     9,0,FW_icon,NULL},
                                                            10,0,FW_icon,NULL},
             {"frm compare",
                                 "compare",
             {"frm_view", NULL,
0,1,FW view,(String)view},
             {"frm_label", video-> name,
                                                     0,12,FW_label,NULL},
             {"frm colors",
                                 "colors",
                                                            13,12,FW_icon,NULL},
      };
```

```
Selection
                   sel = (Selection)MALLOC(sizeof(SelectItem));
      MenuItem
                   trans_menu[TRANS_MENU] = {
             {"trans Wavelet", smeBSBObjectClass, "Wavelet", NULL}.
      };
      Menultem
                   comp_menu[COMP MENU] = {
             {"comp KLICS", smeBSBObjectClass, "KLICS", NULL},
            {"comp KLICS SA", smeBSBObjectClass, "KLICS SA", NULL},
      };
      XtCallbackRec
                         frame call[] = {
             {FrameDestroy,(caddr_t)frame}, {Free,(caddr_t)sel}, {NULL,NULL},
             {CopyVideo,(caddr_t)video}, {NULL,NULL},
             {Examine,(caddr t)frame}, {NULL, NULL}.
             {FramePointYN,(caddr t)frame}, {NULL,NULL},
             {FrameInfo,(caddr t)frame}, {NULL,NULL},
            {FrameMerge,(caddr_t)frame}, {NULL,NULL},
             {Movie,(caddr_t)frame}, {NULL,NULL},
            {PostScript,(caddr_t)frame}, {NULL,NULL},
            {Select,(caddr_t)sel}, {NULL,NULL},
            {Spectrum,(caddr_t)frame}, {NULL,NULL},
      , image_call[ = {
            {NewPoint,(caddr_t)frame}, {NULL,NULL},
      }, trans call[]={
            {Transform,(caddr_t)video}, {NULL,NULL},
      }, comp call[]={
            {Compress,(caddr t)video}, {NULL, NULL}.
            {KlicsSA,(caddr t)video}, {NULL,NULL},
      };
                  cmap = ChannelCmap(frame-> channel = (video-> type = = MONO
      Colormap
| video-> trans.type! = TRANS_None)?0:3, video-> type, video-> gamma);
      Dprintf("InitFrame\n");
```

```
sel->name="video_Compare";
sel->button="frm_compare";
sel->list proc=VideoCurrentList;
sel->action_name = "Compare videos";
sel->action_proc=Compare;
sel->action_closure=(caddr t)video;
frame-> video = video;
frame-> shell = ShellWidget("frm_shell",global-> toplevel,SW_top,cmap,NULL);
form=FormatWidget("frm_form", frame-> shell);
frame->image widget=NULL;
frame-> msg=NULL;
frame-> zoom=0;
frame-> frame=0;
frame->point_switch=False;
frame->point_merge=False;
frame->point=(Point)MALLOC(sizeof(PointRec));
frame-> point-> location[0] = 0;
frame->point->location[1]=0;
frame-> point-> usage = 1;
frame->point->next=global->points;
global->points=frame->point;
frame->palette=0;
frame-> next = global-> frames;
global-> frames = frame;
GetFrame(video, frame-> frame);
```

}

{

```
pixmap = UpdateImage(frame);
       FillForm(form,FRAME_ICONS,items,widgets,frame_call);
       shell[0] = ShellWidget("frm_trans_menu", widgets[4], SW_menu, NULL, NULL);
      FillMenu(shell[0],TRANS_MENU,trans_menu,trans_widgets,trans_call);
      shell[1] = ShellWidget("frm_comp_menu", widgets[7], SW_menu, NULL, NULL);
      FillMenu(shell[1],COMP_MENU,comp_menu,comp_widgets,comp_call);
      frame->point_merge_widget=widgets[6];
      XtSetArg(args[0], XtNbitmap, pixmap);
      XtSetArg(args[1], XtNwidth, video-> size[0]);
      XtSetArg(args[2], XtNheight, video-> size[1]);
      XtSetArg(args[3], XtNcallback, image call);
frame-> image_widget = XtCreateManagedWidget("frm_image", imageWidgetClass, widget
s[11], args, FOUR);
      XtSetSensitive(frame-> image_widget,False);
      XtSetSensitive(widgets[13], PseudoColor = = global -> visinfo-> class):
      XtPopup(frame-> shell, XtGrabNone);
Video FindVideo(name, video)
String name;
Video video;
      if (video = NULL) return(NULL);
      else if (!strcmp(name, video- > name)) return(video);
             else return(FindVideo(name, video-> next));
```

source/InitMain.c

```
/+
       Initialise menu structure for Main command widget
*/
#include
          "../include/xwave.h"
/* Save externs */
             VideoSave();
extern void
             VideoXimSave();
extern void
             VideoDTSave();
extern void
extern void
             VideoMacSave();
              VideoHexSave();
extern void
/* List externs */
extern String *VideoList();
extern String *VideoDropList();
extern String *VideoCurrentList();
extern String *KlicsList();
extern String *KlicsListSA();
/* Import externs */
extern void
             ImportKlics();
extern void ImpKlicsTestSA();
/* Main externs */
```

```
extern void
             Select();
             VideoClean();
extern void
extern void
             Quit();
             VideoLoad();
extern void
extern void
             InitFrame();
             VideoDrop();
extern void
extern void
             PlotGraph();
      Function Name:
                           InitMain
      Description:
                    Create main menu button & sub-menus
      Arguments:
                    none
      Returns:
                    none
 */
#define :
             MAIN_MENU
#define
             SAVE MENU
                                 5
#define
             IMPT_MENU
                                 2
InitMain()
{
      Widget
                    form = FormatWidget("xwave_form", global->toplevel), widgets[1],
                    main_shell, main_widgets[MAIN_MENU],
                    save_shell, save_widgets[SAVE_MENU],
                    impt_shell, impt_widgets[IMPT_MENU];
      FormItem
                   items[] = {
             {"xwaveLogo", "main", 0, 0, FW_icon_button, "xwave_main_sh"},
      };
      Menultem
                   main_menu[] = {
             {"main Open", smeBSBObjectClass, "Open a video", NULL},
             {"main_Attach", smeBSBObjectClass, "Attach a frame", NULL},
             {"main_Save", smeBSBprObjectClass, "Save a video", "xwave save sh"},
```

```
{"main Drop", smeBSBObjectClass, "Drop a video", NULL}.
             {"main_Clean", smeBSBObjectClass, "Clean out videos", NULL}.
             {"main_Import", smeBSBprObjectClass, "Import a
video", "xwave_impt sh"},
             {"main Quit", smeBSBObjectClass, "Quit", NULL}.
      \}, save menu[]={
             {"save menu_vid", smeBSBObjectClass, "Save xwave video", NULL}.
             {"save_menu_xim", smeBSBObjectClass, "Save xim video", NULL}.
             {"save_menu_dt",smeBSBObjectClass, "Save DT image", NULL},
             {"save menu mac", smeBSBObjectClass, "Save mac video", NULL},
             {"save_menu_hex", smeBSBObjectClass, "Save hex dump", NULL}.
      \}, impt menu[]={
             {"impt menu klics", smeBSBObjectClass, "KLICS", NULL}.
             {"impt_menu_klicsSA",smeBSBObjectClass, "KLICS SA", NULL},
      };
      static SelectItem
                           selection[] = {
             {"video_Open", "xwaveLogo", VideoList, "Open a
video", VideoLoad, NULL},
             {"frame_Attach", "xwaveLogo", VideoCurrentList, "Attach a
frame", InitFrame, NULL},
             {"video Drop", "xwaveLogo", VideoDropList, "Drop a
video", Video Drop, NULL},
      \ \, save sel[]={
             {"save vid", "xwaveLogo", VideoCurrentList, "Save xwave
video", VideoSave, NULL},
             {"save xim", "xwaveLogo", VideoCurrentList, "Save xim
video", VideoXimSave, NULL},
             {"save_dt", "xwaveLogo", VideoCurrentList, "Save DT
image", Video DTS ave, NULL \}.
             {"save mac", "xwaveLogo", VideoCurrentList, "Save mac
video", Video MacSave, NULL},
             {"save hex", "xwaveLogo", VideoCurrentList, "Save hex
```

WO 94/23385

- 182 -

```
dump", VideoHexSave, NULL}.
      {"impt_klics", "xwaveLogo", KlicsList, "Import
KLICS", ImportKlics, NULL},
             {"impt_klicsSA", "xwaveLogo", KlicsListSA, "Import KLICS
SA", ImpKlicsTestSA, NULL}.
      };
      XtCallbackRec
                          main call =
             {Select,(caddr_t)&selection[0]}, {NULL,NULL},
             {Select,(caddr t)&selection[1]}, {NULL,NULL},
             {Select,(caddr t)&selection[2]}, {NULL,NULL},
             {VideoClean,(caddr_t)NULL}, {NULL,NULL},
             {Quit,(caddr_t)NULL}, {NULL,NULL},
      }, save call[]={
             {Select,(caddr_t)&save sel[0]}, {NULL,NULL},
             {Select,(caddr_t)&save_sel[1]}, {NULL,NULL},
             {Select,(caddr_t)&save_sel[2]}, {NULL,NULL},
             {Select,(caddr_t)&save_sel[3]}, {NULL,NULL},
             {Select,(caddr_t)&save sel[4]}, {NULL,NULL},
      , impt_call[] = {
             {Select,(caddr_t)&impt_sel[0]}, {NULL,NULL},
             {Select,(caddr_t)&impt_sel[1]}, {NULL,NULL},
      };
      Dprintf("InitMain\n");
      FillForm(form, ONE, items, widgets, NULL):
      main_shell=ShellWidget("xwave_main_sh", widgets[0], SW menu, NULL, NULL);
      save_shell=ShellWidget("xwave_save_sh",main_shell,SW_menu,NULL,NULL);
      impt_shell=ShellWidget("xwave_impt_sh",main_shell,SW memu,NULL,NULL);
      FillMenu(main_shell,MAIN_MENU,main_menu,main_widgets,main_call);
      FillMenu(save shell, SAVE_MENU, save_menu, save_widgets, save_call);
      FillMenu(impt_shell,IMPT_MENU,impt_menu,impt_widgets,impt_call);
}
```

source/Klics5.c

```
/*
       Full still/video Knowles-Lewis Image Compression System utilising HVS
properties
       and delta-tree coding
*/
#include
           "xwave.h"
#include
              "Klics.h"
#include
              < math.h>
             bopen();
extern Bits
             bclose(), bread(), bwrite(), bflush();
extern void
extern WriteKlicsHeader();
/* token modes (empty) */
#define
             EMPTY
#define
             CHANNEL_EMPTY
#define
             OCTAVE_EMPTY
#define
             LPF_EMPTY
#define FULL
typedef
             struct _HistRec
                                  {
             bits, octbits[3][5], lpf, activity, target, token[TOKENS], coeff[129];
      double
                    q const;
} HistRec, *Hist; /* history record */
      Function Name:
                           Access
      Description: Find index address from co-ordinates
```

- 184 -

```
Arguments: x, y - (x,y) co-ordinates
                            oct, sub, channel - octave, sub-band and channel co-ordinates
                            width - image data width
       Returns: index into vid->data[channel][][index]
 */
       Access(x,y,oct,sub,width)
int
int
       x, y, oct, sub, width;
{
       return(((x < <1)+(sub > >1)+width*((y < <1)+(1&sub))) < < oct);
}
       Function Name:
                            LastFrame
       Description: Find last frame encoded
       Arguments: z - index of current frame
                            hist - history records
       Returns:
                     index of previous frame
+/
int
       LastFrame(z,hist)
int
       z;
Hist
      hist;
{
       int
             i=z-1;
       while(hist[i].bits = =0 && i > 0) i--;
       return(i < 0?0:i);
}
```

```
/*
      Function Name:
                          Decide
      Description: Calculate value representing the difference between new and old
blocks
      Arguments:
                  new, old - blocks to compare
                          mode - differencing algorithm {MAXIMUM | SIGABS |
SIGSQR}
      Returns:
                    difference value
 */
int
      Decide(new,old,mode)
Block new, old;
int
      mode;
      int
             X, Y, sigma = 0;
      for(X=0;X<BLOCK;X++) for(Y=0;Y<BLOCK;Y++) {
                   n_o = new[X][Y] - old[X][Y];
             int
             switch(mode) {
             case MAXIMUM:
                   sigma = sigma > abs(n_o)?sigma:abs(n_o);
                   break;
             case SIGABS:
                   sigma + = abs(n_o);
                   break;
             case SIGSQR:
                   sigma + = n_o * n_o;
                   break;
             }
      }
```

```
return(sigma);
}
/*
       Function Name:
                            DecideDouble
       Description: Calculates normal w.r.t differencing algorithm
       Arguments: norm - normal value
                           mode - differencing algorithm {MAXIMUM | SIGABS |
SIGSQR}
       Returns:
                    new normal value
 */
             DecideDouble(norm, mode)
double
double
             norm;
int
       mode;
{
       double
                    ret;
       switch(mode) {
       case MAXIMUM:
             ret = norm;
             break;
      case SIGABS:
             ret = 4.0 * norm;
             break;
      case SIGSQR:
             ret = 4.0 * norm * norm;
             break;
      return(ret);
}
```

WO 94/23385

- 187 -

```
Boolean
              Decision(new,old,norm,mode)
Block new, old;
double
              norm;
int
       mode;
{
       return((double)Decide(new,old,mode) < = DecideDouble(norm,mode));
}
/*
       Function Name:
                             Feedback
       Description: Calculates new target activity from target bits and historical values
       Arguments:
                     hist - history records
                             curr - current frame
                             taps - size of history window
       Returns:
                     target activity
 */
int
       Feedback(hist,curr,taps)
int
       curr;
Hist
       hist:
int
       taps;
{
       int
              prev=curr, i;
       double
                      ratio=0;
       for(i=0; i < taps && prev! = 0; i++)
               prev=LastFrame(prev,hist);
ratio + = (double)hist[prev].activity/(double)(hist[prev].bits-(prev = = 0?hist[0].lpf:0));
```

```
return((int)(ratio*(double)hist[curr].target/(double)i));
}
       Function Name:
                              Filter
       Description: Calculates new q_const filtering historical values
       Arguments:
                      hist - history records
                             curr - current frame
                             taps - size of history window
                             filter - index to filter
       Returns:
                      q_const
              Filter(hist, curr, taps, filter)
double
int
       curr;
Hist
       hist;
       taps, filter;
int
{
       double
                      mac=hist[curr].q_const, sum=1.0, coeff=1.0;
       int
              i, prev=curr;
       for(i=0; i < taps && prev! = 0; i++) {
              prev = LastFrame(prev,hist);
              coeff = filter = = 0?0:coeff/2.0;
              mac + = hist[prev].q_const*coeff;
              sum + = coeff;
       }
       return(mac/sum);
}
```

```
/*
       Function Name: Huffman
       Description: Calculates the number of bits for the Huffman code representing
level
       Arguments: level - level to be encoded
       Returns:
                     number of bits in codeword
 */
       Huffman(level)
int
int
       level:
{
       return(level = 0.2:(abs(level) < 3.2:1 + abs(level)));
/*
       Function Name:
                           HuffCode
       Description: Generates Huffman code representing level
       Arguments:
                    level - level to be encoded
       Returns:
                    coded bits in char's
*/
unsigned char *HuffCode(level)
int
       level;
{
      unsigned char *bytes=(unsigned char *)MALLOC((7+Huffman(level))/8);
       bytes[0] = (abs(level) < 3?abs(level):3) | (level < 0?4:0);
       if (abs(level) > 2) {
                    index = (7 + Huffman(level))/8-1;
             int
```

```
- 190 -
```

```
bytes[index] = bytes[index] | (1 < (Huffman(level)-1)\%8);
       return(bytes);
}
unsigned char *CodeInt(number, bits)
int
       number, bits;
{
       int
              len=(7+bits)/8;
       unsigned char *bytes=(unsigned char *)MALLOC(len);
       int
              byte;
       for(byte=0;byte<len;byte++) {
              bytes[byte] = 0xff&number;
              number = number > > 8;
       return(bytes);
}
       ReadInt(bits,bfp)
int
       bits;
int
Bits
       bfp;
{
       int
              len=(7+bits)/8;
      unsigned char bytes[len];
              byte, number=0;
       int
       bread(bytes, bits, bfp);
```

```
for(byte = 0; byte < len; byte + +)
               number = number | ((int)bytes[byte] < < byte*8);
       number = (number < < sizeof(int)*8-bits) > > sizeof(int)*8-bits;
       return(number);
}
/*
       Function Name:
                             HuffRead
       Description: Read Huffman encoded number from binary file
       Arguments:
                      bfp - binary file pointer
       Returns:
                      decoded level
 */
int
       HuffRead(bfp)
Bits
       bfp;
{
       int
              value;
       unsigned char
                             byte;
       Boolean
                      negative = False;
       bread(&byte,2,bfp);
       value = (int)byte;
       if (byte = = '\0') return(0);
       else {
              bread(&byte,1,bfp);
              negative = (byte! = '\0');
       if (value < 3) return(negif(negative, value));
       for(byte='\0';byte=='\0';value++) bread(&byte,1,bfp);
       return(negif(negative, value-1));
}
```

```
/*
       Function Name:
                            Ouantize
       Description: RM8 style quantizer
       Arguments:
                     data - unquantised number
                            q - quantizing divisor
                            level - quantised to level
                     quantized data & level
       Returns:
 */
int
       Quantize(data,q,level)
int
      data, q, *level;
{
       int
             mag_level = abs(data)/q;
       *level=negif(data<0,mag_level);
      return(negif(data < 0, mag_level*q+(mag_level! =0?(q-1) > 1:0)));
}
/*
      Function Name:
                            Proposed
      Description: Calculates proposed block values
      Arguments: pro - proposed block
                            lev - proposed block quantized levels
                            old, new - old and new block values
                           decide - decision algorithm
                            norms - HVS normals
                    new = 0, proposed values (pro) and levels (lev)
      Returns:
*/
Boolean
             Proposed(pro,lev,old,new,decide,norms)
Block pro, lev, old, new;
```

```
decide;
int
double
             norms[3];
{
      Block zero_block = \{\{0,0\},\{0,0\}\};
      int
             X, Y, step = norms[0] < 1.0?1:(int)norms[0];
                    zero = Decision(new,zero_block,norms[1],decide);
      Boolean
       for(X=0;X < BLOCK;X++) for(Y=0;Y < BLOCK;Y++)
pro[X][Y] = zero?0:old[X][Y] + Quantize(new[X][Y]-old[X][Y], step, &(lev[X][Y]));
      return(zero);
}
                           ZeroCoeffs
       Function Name:
       Description: Zero out video data
       Arguments:
                    data - image data
                           addr - addresses
                    zeros data[addr[][]]
       Returns:
 */
       ZeroCoeffs(data,addr)
void
short *data;
Block addr;
              X, Y;
       int
       for(X=0;X < BLOCK;X++) for(Y=0;Y < BLOCK;Y++)
              data[addr[X][Y]] = 0;
}
```

```
/*
      Function Name:
                           BlockZero
      Description: Test if all block values are zero
      Arguments: block - block under test
      Returns:
                    block = = 0
 */
Boolean
             BlockZero(block)
Block block;
{ .
      int
             X. Y:
      Boolean
                   zero = True;
      for(X=0;X < BLOCK;X++) for(Y=0;Y < BLOCK;Y++)
             if (block[X][Y]!=0) zero=False;
      return(zero);
}
/*
      Function Name:
                          SendToken
      Description: Increments token frequency
      Arguments: token - token to be transmitted
                          channel, sub, oct - co-ordinates
                          ctrl - control record for compresssion
                          hist - history record
                          empty - zero state {EMPTY | CHANNEL EMPTY |
OCTAVE_EMPTY | LPF_EMPTY | FULL}
                          branch - branch of tree (0-3)
      Returns:
                   encodes token
*/
void SendToken(token,channel,sub,oct,ctrl,hist,empty,branch)
```

```
token, channel, sub, oct, *empty, branch:
int
CompCtrl
            ctri;
      hist;
Hist
{
            full=FULL, i;
      int
      String
token_name[TOKENS] = {"ZERO_STILL","NON_ZERO_STILL","BLOCK_SAME","ZE
RO VID", "BLOCK CHANGE",
"LOCAL_ZERO", "LOCAL_NON_ZERO", "CHANNEL_ZERO", "CHANNEL_NON ZE
RO", "OCT ZERO", "OCT NON ZERO",
"LPF_ZERO", "LPF_NON_ZERO", "LPF_LOC ZERO", "LPF LOC NON ZERO" \}:
      switch(*empty) {
      case EMPTY:
            if (token! = ZERO_STILL && token! = BLOCK SAME) {
SendToken(LOCAL_NON_ZERO,channel,sub,oct,ctrl,hist,&full,branch);
                  for(i=0; i < channel; i++)
SendToken(CHANNEL ZERO, i, sub, oct, ctrl, hist, & full, branch);
                  *empty = CHANNEL_EMPTY;
                  SendToken(token,channel,sub,oct,ctrl,hist,empty,branch);
            }
            break:
      case CHANNEL_EMPTY:
            if (token! = ZERO_STILL && token! = BLOCK SAME) {
SendToken(CHANNEL_NON_ZERO,channel,sub,oct,ctrl,hist,&full,branch):
                  for(i=1;i < sub;i++)
SendToken(token = = NON_ZERO_STILL?ZERO_STILL:BLOCK_SAME,channel,i,oct,ct
```

```
ri, hist, & full, branch);
                    *empty = FULL;
                    SendToken(token,channel,sub,oct,ctrl,hist,empty,branch);
             }
             break;
      case OCTAVE EMPTY:
             if (token! = ZERO STILL && token! = BLOCK SAME) {
SendToken(OCT NON ZERO, channel, sub, oct, ctrl, hist, & full, branch);
                    for(i=0; i < branch; i++)
SendToken(token = = NON ZERO STILL?ZERO_STILL:BLOCK SAME,channel,sub,oc
t,ctrl,hist,&full,branch);
                    *empty = FULL;
                    SendToken(token,channel,sub,oct,ctrl,hist,empty,branch);
             }
             break:
      case LPF EMPTY:
             if (token!=LPF_ZERO) {
SendToken(LPF_LOC_NON_ZERO,channel,sub,oct,ctrl,hist,&full,branch);
                    for(i=0; i < channel; i++)
SendToken(LPF ZERO, i, sub, oct, ctrl, hist, & full, branch);
                    *empty = FULL;
                    SendToken(token,channel,sub,oct,ctrl,hist,empty,branch);
             break;
       case FULL:
             Dprintf("%s\n",token name[token]);
             hist->token[token]++;
             hist->bits+=token bits[token];
             hist->octbits[channel][oct] + = token_bits[token];
             if (ctrl->bin_switch)
```

```
bwrite(&token_codes[token],token bits[token],ctrl->bfp);
              break;
       }
}
       Function Name:
/*
                            ReadBlock
       Description: Read block from video
       Arguments:
                     new, old, addr - new and old blocks and addresses
                            x, y, z, oct, sub, channel - co-ordinates of block
                            ctrl - compression control record
       Returns:
                     block values
 */
       ReadBlock(new,old,addr,x,y,z,oct,sub,channel,ctrl)
Block new, old, addr;
       x, y, z, oct, sub, channel;
CompCtrl
             ctrl;
{
              X, Y;
       int
       for(X=0;X<BLOCK;X++) for(Y=0;Y<BLOCK;Y++) {
addr[X][Y] = Access((x < < 1) + X, (y < < 1) + Y, oct, sub, Size(ctrl-> src, channel, 0));
              new[X][Y] = (int)ctrl-> src-> data[channel][z][addr[X][Y]];
              old[X][Y] = (int)ctrl-> dst-> data[channel][z][addr[X][Y]];
       }
}
/*
       Function Name:
                            CalcNormals
       Description: Calculates HVS weighted normals
```

```
Arguments: ctrl - compression control record
                           oct, sub, channel - co-ordinates
                           norms - pre-initialised normals
                    weighted normals
       Returns:
*/
      CalcNormals(ctrl,oct,sub,channel,norms)
void
CompCtrl
             ctrl;
       oct, sub, channel;
int
double
             norms[3];
{
       Video vid=ctrl->dst;
             norm, base oct=oct+(vid->type==YUV \&\&
channel! = 0?vid-> trans. wavelet.space[0]-vid-> trans. wavelet.space[1]:0) + (sub = = 0?1:0)
       for(norm=0;norm<3;norm++) {
             if (norm! = 0) norms[norm] *= ctrl-> quant const;
             norms(norm) *=
ctrl->base_factors[base_oct]*(sub = = 3?ctrl-> diag_factor: 1.0);
             if (channel! = 0) norms[norm] *= ctrl->chrome factor;
             norms[norm] *=(double)(1 < vid-> precision);
       }
}
                           MakeDecisions
       Function Name:
/*
       Description: Decide on new compression mode from block values
       Arguments: old, new, pro - block values
                           zero - zero flag for new block
                           norms - HVS normals
```

```
mode - current compression mode
                           decide - comparison algorithm
                    new compression mode
      Returns:
 */
int
      MakeDecisions(old,new,pro,zero,norms,mode,decide)
Block new, old, pro;
Boolean
             zero;
double
             norms[3];
      mode, decide;
int
{
      Block zero_block = \{\{0,0\},\{0,0\}\};
             new_mode, np = Decide(new,pro,decide), no = Decide(new,old,decide);
       int
       if (np < no && (double)no > DecideDouble(norms[mode = = STILL?1:2],decide)
&& !zero)
             new mode=mode==STILL ||
(double)Decide(old,zero_block,decide) < = DecideDouble(norms[1],decide)?STILL:SEND;
       else new_mode=mode==SEND && np<no && zero?VOID:STOP;
       return(new_mode);
}
int
       MakeDecisions2(old,new,pro,lev,zero,norms,mode,decide)
Block new, old, pro, lev;
Boolean
             zero;
double
             norms[3];
       mode, decide;
int
 {
```

```
Block zero_block = \{\{0,0\},\{0,0\}\}\;
       int
             new_mode = mode = = STILL | BlockZero(old)?STILL:SEND.
                    np = Decide(new,pro,decide), no = Decide(new,old,decide);
       if (new_mode = STILL) new_mode = np > = no || zero ||
BlockZero(lev)?STOP:STILL;
      else new_mode=zero && np<no?VOID:np>=no ||
Decision(new,old,norms[2],decide) | BlockZero(lev)?STOP:SEND;
      return(new mode);
}
      Function Name:
                           UpdateCoeffs
      Description: Encode proposed values and write data
       Arguments:
                    pro, lev, addr - proposed block, levels and addresses
                           z, channel, oct - co-ordinates
                           ctrl - compression control record
                           hist - history record
      Returns:
                    alters ctrl->dst->data[channel][z][addr[][]]
 */
void
      UpdateCoeffs(pro,lev,addr,z,channel,oct,ctrl,hist)
Block pro, lev, addr;
int
      z, channel, oct;
CompCtrl ctrl;
Hist
      hist;
{
             X, Y;
      int
      for(X=0;X < BLOCK;X++) for(Y=0;Y < BLOCK;Y++) 
                    bits = Huffman(lev[X][Y]),
             int
```

```
level = abs(lev[X][Y]);
             ctrl-> dst-> data[channel][z][addr[X][Y]] = (short)pro[X][Y];
             hist-> coeff[level > 128?128:level] + +;
             hist-> bits + = bits;
             hist->octbits[channel][oct] + = bits;
             if (ctrl->bin_switch) {
                     unsigned char
                                          *bytes = HuffCode(lev[X][Y]);
                    bwrite(bytes,bits,ctrl->bfp);
                     XtFree(bytes);
              }
      }
                            SendTree
      Function Name:
      Description: Encode tree blocks
      Arguments:
                    prev mode - compression mode
                            x, y, z, oct, sub, channel - co-ordinates
                            ctrl - compression control record
                            hist - history records
                            empty - token mode
                            branch - tree branch number
                     active block indicator
      Returns:
*/
              SendTree(prev_mode,x,y,z,oct,sub,channel,ctrl,hist,empty,branch)
Boolean
      prev_mode, x, y, z, oct, sub, channel, *empty, branch;
int
CompCtrl
              ctrl:
Hist
      hist:
```

```
{
      Block addr, old, new, pro, lev;
             new mode, X, Y;
      int
      double
norms[3] = {ctrl-> quant_const,ctrl-> thresh_const,ctrl-> cmp_const}; /* quant, thresh,
compare */
       Boolean
                    active = False;
       ReadBlock(new,old,addr,x,y,z,oct,sub,channel,ctrl);
       if (prev_mode! = VOID) {
             Boolean
                           zero:
             CalcNormals(ctrl,oct,sub,channel,norms);
             zero = Proposed(pro,lev,old,new,ctrl-> decide,norms);
/*
new mode = MakeDecisions(old, new, pro, zero, norms, prev mode, ctrl-> decide); */
new mode = MakeDecisions2(old, new, pro, lev, zero, norms, prev mode, ctrl-> decide);
             switch(new_mode) {
             case STOP:
/*SendToken(prev_mode = = STILL?ZERO_STILL:BLOCK_SAME,channel,sub,oct,ctrl,h
ist,empty,branch);*/
                    SendToken(prev_mode = = STILL | |
BlockZero(old)?ZERO STILL:BLOCK SAME, channel, sub, oct, ctrl, hist, empty, branch);
                    break:
             case STILL:
             case SEND:
                    active = True;
/*SendToken(prev_mode = = STILL?NON_ZERO_STILL:BLOCK_CHANGE,channel,sub
,oct,ctrl,hist,empty,branch);*/
```

```
SendToken(prev_mode = = STILL | |
BlockZero(old)? NON\_ZERO\_STILL: BLOCK\_CHANGE, channel, sub, oct, ctrl, hist, empty,
branch):
                    UpdateCoeffs(pro,lev,addr,z,channel,oct,ctrl,hist);
                    break:
             case VOID:
                    SendToken(ZERO VID, channel, sub, oct, ctrl, hist, empty, branch);
                    ZeroCoeffs(ctrl->dst->data[channel][z],addr);
                    break;
       } else {
             if (BlockZero(old)) new_mode=STOP;
             else {
                    ZeroCoeffs(ctrl->dst->data[channel][z],addr);
                    new mode=VOID;
              }
       }
       if (oct > 0 && new_mode! = STOP) {
                    mt=OCTAVE_EMPTY, full=FULL;
              int
              Dprintf(x = %d, y = %d, oct = %d sub = %d mode
 %d\n",x,y,oct,sub,new_mode);
              for(Y=0;Y<2;Y++) for(X=0;X<2;X++)
(void) Send Tree (new\_mode, x*2+X, y*2+Y, z, oct-1, sub, channel, ctrl, hist, \&mt, X+2*Y); \\
              if (mt = = OCTAVE_EMPTY && new_mode! = VOID)
 SendToken(OCT_ZERO,channel,sub,oct,ctrl,hist,&full,0);
       return(active);
 }
                            SendLPF
 /*
        Function Name:
```

```
Description: Encode LPF sub-band
                    mode - compression mode
       Arguments:
                           z -
                                  frame number
                           ctrl - compression control record
                           hist - history records
                    encodes data
      Returns:
void
      SendLPF(mode,z,ctrl,hist)
CompCtrl
             ctri:
int
       mode, z;
Hist
      hist:
{
       Block new, old, pro, lev, addr;
             channel, channels=ctrl->src->type==MONO?1:3, x, y, full=FULL,
       int
                    octs_lum = ctrl-> src-> trans.wavelet.space[0],
size[2] = \{Size(ctrl > src, 0, 0) > octs lum + 1, Size(ctrl > src, 0, 1) > octs lum + 1\};
      for(y=0;y < size[1];y++) for(x=0;x < size[0];x++) {
             int
                    empty=LPF_EMPTY;
      for(channel = 0; channel < channels; channel + +) {
                    octs=ctrl->src->trans.wavelet.space[ctrl->src->type==YUV
             int
&& channel! = 0?1:0],
                           new mode, X, Y, step, value, bits=0;
             double
norms[3] = {ctrl-> quant_const,ctrl-> thresh_const,ctrl-> cmp_const};
             CalcNormals(ctrl,octs-1,0,channel,norms);
```

```
step = norms[0] < 1.0?1:(int)norms[0];
             for(bits=0,
value = ((1 < 8 + ctrl - > dst - > precision) - 1)/step; value! = 0; bits + +)
                    value = value > > 1;
             ReadBlock(new,old,addr,x,y,z,octs-1,0,channel,ctrl);
              /* Proposed */
             for(X=0;X < BLOCK;X++) for(Y=0;Y < BLOCK;Y++)
pro[X][Y] = old[X][Y] + Quantize(new[X][Y] - old[X][Y], step, &(lev[X][Y]));
              /* MakeDecisions */
new_mode = mode = = STILL?STILL:Decision(new,old,norms[2],ctrl->decide) | |
BlockZero(lev)?STOP:SEND;
              switch(new_mode) {
              case SEND:
                     SendToken(LPF_NON_ZERO,channel,0,octs,ctrl,hist,&empty,0);
                     UpdateCoeffs(pro,lev,addr,z,channel,octs,ctrl,hist);
              break;
              case STILL:
                     for(X=0;X < BLOCK;X++) for(Y=0;Y < BLOCK;Y++) 
                            ctrl->dst->data[channel][z][addr[X][Y]]=(short)pro[X][Y];
                            hist->bits+=bits;
                            hist->octbits[channel][octs] + = bits;
                            if (ctrl->bin_switch) {
                                   unsigned char *bytes=CodeInt(lev[X][Y],bits);
                                   bwrite(bytes,bits,ctrl->bfp);
                                   XtFree(bytes);
                            }
```

```
}
                    break;
             case STOP:
                    SendToken(LPF\_ZERO, channel, 0, octs, ctrl, hist, \&empty, 0);
                    break:
              }
      if (mode! = STILL && empty = = LPF_EMPTY)
SendToken(LPF_LOC_ZERO,channel,0,octs_lum,ctrl,hist,&full,0);
       hist-> lpf = hist-> bits;
}
                            LookAhead
       Function Name:
       Description: Examine base of tree to calculate new quantizer value
       Arguments: z - frame number
                            ctrl - compression control record
                            hist - history records
       Returns:
                     calculates new ctrl->quant_const
       LookAhead(z,ctrl,hist)
void
CompCtrl
              ctrl;
int
       z;
Hist
       hist:
{
                     x, y, sub, index, thresh[HISTO], decide=ctrl->decide, act,
              int
                            taract=Feedback(hist,z.ctrl-> feedback),
                            octs = ctrl-> src-> trans. wavelet.space[0],
```

```
size[2] = \{Size(ctrl->src,0,0) > 1 + octs, Size(ctrl->src,0,1) > 1 + octs\};
             Block new, old, addr;
                           old quant=ctrl->quant_const;
             double
             ctrl->quant_const=1.0;
             for(index = 0; index < HISTO; index + +) thresh[index] = 0;
             for(y=0;y < size[1];y++) for(x=0;x < size[0];x++)
for(sub = 1; sub < 4; sub + +)
                    double
                                  q_thresh[3],
norms[3] = {ctrl-> quant_const,ctrl-> thresh_const,ctrl-> cmp_const};
                    Block zero block = \{\{0,0\},\{0,0\}\}\;
                    ReadBlock(new,old,addr,x,y,z,octs-1,sub,0,ctrl);
                     CalcNormals(ctrl,octs-1,sub,0,norms);
q thresh[1] = (double)Decide(new,zero_block,decide)/DecideDouble(norms[1],decide);
q thresh[2] = (double)Decide(new,old,decide)/DecideDouble(norms[2],decide);
                     if (BlockZero(old)) q thresh[0]=q thresh[1];
                     clse q_thresh[0] = q_thresh[2] < q_thresh[1]?q_thresh[2]:q_thresh[1];
                     if (ctrl->decide = = SIGSQR) q_thresh[0] = sqrt(q_thresh[0]);
index = (int)((q_thresh[0]-old_quant+HISTO_DELTA)*HISTO/(HISTO_DELTA*2));
                     index = index < 0?0:index > HISTO-1?HISTO-1:index;
                     thresh[index]++;
              }
              for(index=HISTO-1, act=0; index>=0 && act< taract; index--)
act + = thresh[index];
ctrl->quant_const=(double)(index+1)*HISTO_DELTA*2.0/HISTO+old_quant-HISTO_
 DELTA:
              ctrl->quant_const=ctrl->quant_const<0.0?0.0:ctrl->quant_const;
```

```
Dprintf("Target bits %d act %d (real %d) adjust q_const to
%3.2f\n",hist[z].target,taract,act,ctrl->quant_const);
             hist[z].q const=ctrl->quant const;
             ctrl->quant_const=Filter(hist,z,ctrl-> feedback,ctrl-> filter);
             Dprintf("Post filtering q_const to %3.2f\n",ctrl->quant_const);
              if (ctrl->bin_switch) {
                     unsigned char *bytes=CodeInt(index+1-HISTO/2,HISTO_BITS);
                     bwrite(bytes,HISTO_BITS,ctrl->bfp);
                     XtFree(bytes);
}
                            CompressStats
/*
       Function Name:
       Description: Compile compression statistics
                     ctrl - compression control record
                            hist - history records
                     plot graphs
       Returns:
 */
       CompressStats(ctrl,hist)
void
CompCtrl
              ctrl;
Hist
       hist;
{
       FILE *fp_token, *fp_coeff, *fp_log, *fopen();
              file name[STRLEN];
       char
              channel, z, i, sigma;
       int
```

 $sprintf(file_name, "\%s\%s/\%s.token\%s\\ \label{file_name} sprintf(file_name, "\%s\%s/\%s.token\%s)\\ \label{file_name} sprintf(file_name, "\%s\%s/\%s.token\%s)\\ \label{file_name} sprintf(file_name, "\%s\%s/\%s.token\%s)\\ \label{file_name} sprintf(file_name, "\%s\%s, "%s%s, "%s$

```
LOT_EXT);
       fp token = fopen(file_name, "w");
sprintf(file_name, "%s%s/%s.coeff%s\0",global->home,PLOT_DIR,ctrl->stats_name,PL
OT_EXT);
       fp coeff = fopen(file_name, "w");
sprintf(file_name, "%s%s/%s.log%s\0",global->home,PLOT_DIR,ctrl->stats_name,PLO
T EXT);
       fp log = fopen(file_name, "w");
       fprintf(fp_token, "\"Tokens %s\n",ctrl-> name);
       for(i=0;i < TOKENS;i++) {
               sigma = 0;
               for(z=0;z < ctrl-> src-> size[2];z++) sigma+=hist[z]:token[i];
               fprintf(fp token, "%d %d\n", i, sigma);
        fprintf(fp_coeff, "\"Coeffs %s\n",ctrl->name);
        for(i=0; i < 129; i++)
               sigma = 0;
               for(z=0;z < ctrl-> src-> size[2];z++) sigma+=hist[z].coeff[i];
               fprintf(fp coeff, "%d %d\n", i, sigma);
        }
        for(i=0;i<5;i++) {
               String titles[5] = {"treebits", "activity", "quant", "bits", "ratio"};
               fprintf(fp log,"\n\"%s\n",titles[i]);
               for(z=0;z < ctrl-> src-> size[2];z++)
                      switch(i) {
                      case 0: fprintf(fp_log, "%d %d\n", z, hist[z].bits-hist[z].lpf);
                                    break:
                       case 1: fprintf(fp_log, "%d %d\n", z, hist[z].activity);
                                    break;
```

```
case 2: fprintf(fp_log, "%d %f\n",z,hist[z].q_const);
                                    break:
                                    fprintf(fp log, "%d %d\n", z, hist[z].bits);
                     case 3:
                                   break;
                                    fprintf(fp_log, "%d
                     case 4:
%f\n^{-},z,(double)(hist[z].bits-(z==0?hist[z].lpf:0))/(double)hist[z].activity);
                                    break;
                      }
       }
       for(channel = 0; channel < (ctrl-> src-> type = = MONO?1:3); channel + +) {
                     octs = ctrl-> src-> trans. wavelet. space[ctrl-> src-> type = = YUV
&& channel! = 0?1:0];
       for(i=0; i < = octs; i++) 
              fprintf(fp_log,"\n\"channel %d oct %d\n",channel,i);
              for(z=0;z < ctrl-> src-> size[2];z++)
                      fprintf(fp_log, "%d %d\n",z,hist[z].octbits[channel][i]);
       }
       fclose(fp_token); fclose(fp_coeff); fclose(fp_log);
}
       Function Name:
                             CopyFrame
/*
       Description: Copy frame or zero
        Arguments:
                      vid - video
                              from, to - source and destination frame numbers
                              zero - zero out flag
                      alters video->data
        Returns:
  */
        CopyFrame(vid,from,to,zero)
 void
```

```
Video vid;
int
      from, to;
Boolean
             zero;
{
              i, channel;
       int
       for(channel=0;channel<(vid->type==MONO?1:3);channel++) {
                     size = Size(vid,channel,0)*Size(vid,channel,1);
              int
              for(i=0; i < size; i++)
                     vid->data[channel][to][i] = zero?0:vid->data[channel][from][i];
       }
}
       Function Name:
                            CompressFrame
       Description:
                     Compress a Frame
       Arguments:
                     ctrl - compression control record
                            z - frame number
                            hist - history records
                            target - target bits
 */
       CompressFrame(ctrl,z,hist,target)
void
CompCtrl
              ctrl;
int
        z, target;
Hist
       hist:
{
        Video src=ctrl->src, dst=ctrl->dst;
              sub, channel, x, y, mode=ctrl-> stillvid | | z = 0?STILL:SEND,
        int
```

```
octs lum=src->trans.wavelet.space[0],
size[2] = \{Size(src,0,0) > 1 + octs lum, Size(src,0,1) > 1 + octs lum\};
      NewFrame(dst,z);
      CopyFrame(dst, z-1, z, ctrl-> stillvid ||z==0\rangle;
      GetFrame(src,z);
      hist[z].target = target;
      if (z! = 0 \&\& ctrl-> auto q) LookAhead(z,ctrl,hist);
      SendLPF(mode, z, ctrl, &hist[z]);
      Dprintf("LPF bits %d\n",hist[z].lpf);
      hist[z].q const=ctrl->quant const;
      for(y=0;y < size[1];y++) for(x=0;x < size[0];x++) {
             int
                    empty=EMPTY, full=FULL,
             for(channel = 0; channel < (dst-> type = = MONO?1:3); channel + +) {
                           octs=src->trans.wavelet.space[src->type==YUV &&
                    int
channel! = 0?1:0];
                    for(sub=1; sub < 4; sub + +) {
                           Boolean
active = SendTree(mode,x,y,z,octs-1,sub,channel,ctrl,&hist[z],&empty,0);
                           hist[z].activity + = channel = = 0 && active;
                    switch(empty) {
                    case FULL:
                           empty = CHANNEL_EMPTY;
                           break;
                    case CHANNEL EMPTY:
                           SendToken(CHANNEL_ZERO,channel,sub,octs-1,ctrl,&hist[z],&full,0)
                           break;
```

```
}
             if (empty = = EMPTY)
SendToken(LOCAL_ZERO,channel,sub,octs_lum-1,ctrl,&hist[z],&full,0);
      Dprintf("Activity: %d\n",hist[z].activity);
      FreeFrame(src,z);
}
       Function Name:
                           SkipFrame
      Description: Shuffle frame data as if current frame was skipped
       Arguments: vid - video
                           z - frame number
       Returns:
                    alters vid->data
void
      SkipFrame(vid,z)
Video vid;
int
      z;
{
      NewFrame(vid,z);
      CopyFrame(vid,z-1,z,False);
      if (z > 1) {
             GetFrame(vid,z-2);
             CopyFrame(vid,z-2,z-1,False);
             FreeFrame(vid,z-2);
.}
       Function Name:
                           CompressCtrl
```

```
Description: Perform KLICS on a video
       Arguments:
                    w - Xaw widget
                           closure - compression control record
                           call_data - NULL
      Returns:
                    compressed video
 */
void
      CompressCtrl(w,closure,call_data)
Widget
caddr_t
             closure, call_data;
{
      CompCtrl
                    ctrl=(CompCtrl)closure;
      int
             sigma_bits, frame_count, z, i, buffer=0, frames=ctrl->src->size[2],
                    bpf in=(64000*ctrl->bitrate)/ctrl->src->rate,
                    bpf_out = (int)((double)(64000*ctrl->bitrate)/ctrl->fps);
      FILE *fopen();
      char
             file_name[STRLEN];
      HistRec
                    hist[frames];
                   msg = NewMessage(NULL, 60);
      Message
      msg->rows=frames>10?11:frames+(frames==1?0:1); msg->cols=30;
      if (global->batch = = NULL) {
             XtCallbackRec
                                 callbacks[] = {
                    {CloseMessage,(caddr_t)msg}, {NULL,NULL},
             };
MessageWindow(FindWidget("frm compress",w),msg,"KLICS",True,callbacks);
      Dprintf("CompressCtrl\n");
```

```
if (ctrl-> src-> type = = YUV &&
(ctrl-> src-> trans.wavelet.space[0]! = ctrl-> src-> trans.wavelet.space[1] + ctrl-> src-> U
Vsample[0] ~~ |~ |~ ctri-> src-> UVsample[0]! = ctri-> src-> UVsample[1])) ~~ \{
              Eprintf("Y-UV octaves mis-matched. Check UV-sample");
              return:
       }
       ctrl-> dst = CopyHeader(ctrl-> src);
       strcpy(ctrl->dst->name,ctrl->name);
       if (ctrl->dst->disk) SaveHeader(ctrl->dst);
       if (ctrl-> bin switch) {
sprintf(file_name, "%s%s/%s%s\0", global->home, KLICS_DIR, ctrl->bin_name, KLICS_
EXT);
             ctrl->bfp=bopen(file name, "w");
             /* Write some sort of header */
             WriteKlicsHeader(ctrl);
      }
      for(z=0;z < frames;z++) {
             hist[z].bits=0;
             hist[z].lpf=0;
             hist[z].activity=0;
             hist[z].target=0;
             for (i=0; i<5; i++) hist[z].octbits[0][i]=0;
             for(i=0;i<5;i++) hist[z].octbits[1][i]=0;
             for(i=0;i<5;i++) hist[z].octbits[2][i]=0;
             for(i=0; i < TOKENS; i++) hist[z].token[i]=0;
             for(i=0; i < 129; i++) hist[z].coeff[i]=0;
             hist[z].q_const=0.0;
      for(z=0;z < frames;z++) {
             if (z==0 || !ctrl-> buf_switch) {
                    CompressFrame(ctrl,z,hist,bpf out);
```

```
buffer = 3200*ctrl-> bitrate + bpf_in;
             } else {
                     Boolean
                                   no_skip;
                     buffer-=bpf_in;
                     buffer = buffer < 0?0:buffer;
                     no skip=buffer<6400*ctrl->bitrate; /* H.261 buffer size */
                     if (ctrl-> bin switch) bwrite(&no_skip,1,ctrl-> bfp);
                     if (no_skip) {
                            CompressFrame(ctrl,z,hist,bpf_out/*+bpf_out/2-buffer*/);
                            buffer + = hist[z].bits;
                     } else SkipFrame(ctrl->dst,z);
              if (z>0) {
                     SaveFrame(ctrl->dst,z-1);
                     FreeFrame(ctrl->dst,z-1);
              Mprintf(msg, "%s%03d: %d
bits\n".ctrl->dst->name,z+ctrl->src->start,hist[z].bits);
              Mflush(msg);
       SaveFrame(ctrl->dst,ctrl->src->size[2]-1);
       FreeFrame(ctrl->dst,ctrl->src->size[2]-1);
       if (ctrl->bin_switch) { bflush(ctrl->bfp); bclose(ctrl->bfp); }
       if (ctrl->stats_switch) CompressStats(ctrl,hist);
       Dprintf("Compression Complete\n");
       sigma_bits=0, frame_count=0;
       for(z=0;z<ctrl->src->size[2];z++) 
              sigma_bits + = hist[z].bits;
              if (hist[z].bits! = 0) frame_count++;
       if (ctrl->buf_switch) {
```

```
Dprintf("Buffer contains %d bits\n",buffer-bpf_in);
              Dprintf("Frame Rate %4.1f
Hz\n^-,(double)(ctrl-> src-> rate+(frame\_count-1))/(double)(ctrl-> src-> size[2]-1));
       }
      if (frames > 1) {
              Mprintf(msg, "Total: %d bits\n", sigma_bits);
              Mflush(msg);
       ctrl->dst->next=global->videos;
       global-> videos = ctrl-> dst;
}
                            BatchCompCtrl
       Function Name:
       Description: Batch interface to CompressCtrl
 */
       BatchCompCtrl(w,closure,call_data)
void
Widget
              closure, call_data;
caddr t
{
                     ctrl=(CompCtrl)closure;
       CompCtrl
       if (ctrl->src==NULL) ctrl->src=FindVideo(ctrl->src_name,global->videos);
       CompressCtrl(w,closure,call_data);
}
                            InitCompCtrl
        Function Name:
       Description: Initialise the compression control record
                     name - name of the source video
        Arguments:
                     compression control record
        Returns:
```

*/

```
InitCompCtrl(name)
CompCtrl
String name;
{
       CompCtrl
                     ctrl = (CompCtrl)MALLOC(sizeof(CompCtrlRec));
              i;
       int
       ctrl->decide = SIGABS;
       ctrl-> feedback = 4;
       ctrl-> filter=0;
       ctrl-> stillvid=True;
       ctrl-> stats_switch = False;
       ctrl-> auto_q = True;
       ctrl->buf_switch=True;
      ctrl->bin_switch=False;
      ctrl-> cmp_const=0.9;
      ctrl->thresh_const=0.6;
      ctrl-> quant_const=8.0;
      ctrl > fps = 30.0;
       ctrl-> bitrate = 1;
       for(i=0;i<5;i++) {
              double
                            defaults[5] = \{1.0, 0.32, 0.16, 0.16, 0.16\};
             ctrl->base_factors[i] = defaults[i];
      }
      ctrl > diag_factor = 1.4142136;
      ctrl->chrome_factor=2.0;
      strcpy(ctrl-> src_name,name);
      strcpy(ctrl-> name,name);
```

```
strcpy(ctrl-> stats name, name);
      strcpy(ctrl->bin_name,name);
      return(ctrl);
}
/*
      Function Name:
                          Compress
      Description: X Interface to CompressCtrl
*/
#define
             COMP_ICONS
                                 25
#define
             VID ICONS 15
      Compress(w,closure,call_data)
void
Widget
             w;
caddr t
            closure, call data;
{
      Video video=(Video)closure;
      CompCtrl
                   ctrl=InitCompCtrl(video-> name);
            i, space=video-> trans.wavelet.space[0]+1;
      int
      NumInput
                   num_inputs = (NumInput)MALLOC(2*sizeof(NumInputRec));
      FloatInput
                   flt_inputs = (FloatInput)MALLOC(6*sizeof(FloatInputRec)),
oct inputs = (FloatInput)MALLOC(space*sizeof(FloatInputRec));
      Message
                   msg = NewMessage(ctrl-> name, NAME_LEN),
                   msg_bin=NewMessage(ctrl->bin_name,NAME_LEN),
                   msg_stats = NewMessage(ctrl-> stats_name, NAME_LEN);
      XtCallbackRec
                          destroy_call[] = {
            {Free,(caddr_t)ctrl},
            {Free,(caddr t)num inputs},
            {Free,(caddr_t)flt_inputs},
```

```
{Free,(caddr_t)oct_inputs},
              {CloseMessage,(caddr_t)msg},
              {CloseMessage,(caddr_t)msg_bin},
              {CloseMessage,(caddr_t)msg_stats},
              {NULL, NULL},
      };
       Widget
                     parent = FindWidget("frm_compress", XtParent(w)),
                     shell=ShellWidget("klics", parent, SW_below, NULL, destroy call),
                     form = Format Widget("klics form", shell),
dec shell=ShellWidget("klics_cng_dec", shell, SW_menu, NULL, NULL), dec widgets[3],
filt shell=ShellWidget("klics_cng_filt", shell, SW_menu, NULL, NULL), filt widgets[2],
                     widgets[COMP_ICONS], vid_widgets[VID_ICONS],
oct_widgets[space*2];
      FormItem
                     items[] = {
              {"klics_cancel", "cancel", 0,0,FW_icon, NULL},
              {"klics_confirm", "confirm", 1, 0, FW_icon, NULL},
              {"klics_title", "Compress a video", 2, 0, FW_label, NULL},
              {"klics_vid_lab", "Video Name: ",0,3,FW_label,NULL},
              {"klics_vid", NULL, 4, 3, FW_text, (String) msg},
             {"klics stats lab", "Statistics: ",0,4,FW_label,NULL},
             {"klics_stats", NULL, 4, 4, FW_yn, (String)&ctrl-> stats_switch},
              {"klics_stats_name", NULL, 7, 4, FW_text, (String) msg_stats},
              {"klics bin lab", "KLICS File: ",0,6,FW label, NULL},
             {"klics_bin", NULL, 4, 6, FW_yn, (String)&ctrl-> bin_switch},
             {"klics_bin_name", NULL, 10,6, FW_text, (String) msg_bin},
             {"klics_dec_lab", "Decision: ",0,9,FW_label,NULL},
             {"klics_dec_btn", "SigmaAbs", 4,9,FW_button, "klics_cng_dec"},
             {"klics qn float", NULL, 0, 12, FW float, (String)&flt_inputs[0]},
```

```
{"klics qn_scroll", NULL, 4, 12, FW scroll, (String)&flt_inputs[0]},
       {"klics_th_float", NULL, 0, 14, FW_float, (String)&flt_inputs[1]},
       {"klics_th_scroll", NULL, 4, 14, FW_scroll, (String)&flt_inputs[1]},
       {"klics_cm_float", NULL, 0, 16, FW_float, (String)&flt_inputs[2]},
       {"klics cm scroll", NULL, 4, 16, FW scroll, (String)&flt inputs[2]}.
       {"klics ch float", NULL, 0, 18, FW float, (String)&flt inputs[3]},
       {"klics_ch_scroll", NULL, 4, 18, FW_scroll, (String)&flt_inputs[3]},
       {"klics_di_float", NULL, 0, 20, FW_float, (String)&flt_inputs[4]},
       {"klics di scroll", NULL, 4, 20, FW scroll, (String) & fit inputs [4]},
       {"klics oct form", NULL, 0, 22, FW form, NULL},
       {"klics_vid_form", NULL, 0, 24, FW_form, NULL},
, vid items = {
       {"klics_ic_lab", "Image Comp: ",0,0,FW label,NULL},
       {"klics ic", NULL, 1, 0, FW yn, (String)&ctrl-> stillvid},
       {"klics_tg_float", NULL, 0, 1, FW_float, (String)&flt_inputs[5]},
       {"klics tg scroll", NULL, 1, 1, FW_scroll, (String)&flt inputs[5]},
       {"klics_px_int", NULL, 0, 3, FW_integer, (String)&num_inputs[0]},
       {"klics_px_down", NULL, 1, 3, FW_down, (String)&mum_inputs[0]},
       {"klics_px_up", NULL, 6, 3, FW_up, (String)&mum_inputs[0]},
       {"klics_auto_lab", "Auto Quant: ",0,5,FW_label,NULL},
       {"klics_auto", NULL, 1, 5, FW_yn, (String)&ctrl->auto q},
       {"klics buf lab", "Buffer: ",0,8,FW label, NULL},
       {"klics_buf",NULL,1,8,FW_yn,(String)&ctrl->buf_switch},
       {"klics buf btn", "None", 11,8,FW button, "klics cng filt"},
       {"klics_hs_int", NULL, 0, 10, FW_integer, (String)&num_inputs[1]},
       {"klics_hs_down", NULL, 1, 10, FW_down, (String)&num_inputs[1]},
       {"klics hs up", NULL, 14, 10, FW up, (String) & num inputs[1]},
}, oct items[2*space];
```

```
MenuItem
             dec_menu[] = {
      {"klics dec max", smeBSBObjectClass, "Maximum", NULL},
      {"klics_dec_abs", smeBSBObjectClass, "SigmaAbs", NULL},
      {"klics dec sqr", smeBSBObjectClass, "SigmaSqr", NULL},
\}, filt menu[]={
      {"klics_filt_none", smeBSBObjectClass, "None", NULL},
      {"klics_filt_exp",smeBSBObjectClass,"Exp",NULL},
};
XtCallbackRec
                   callbacks[] = {
      {Destroy,(caddr_t)shell},
      {NULL, NULL},
      {CompressCtrl,(caddr t)ctrl},
      {Destroy,(caddr_t)shell},
      {NULL, NULL},
      {ChangeYN,(caddr_t)&ctrl-> stats_switch}, {NULL,NULL},
      {ChangeYN,(caddr_t)&ctrl->bin_switch}, {NULL,NULL},
      {FloatIncDec,(caddr t)&flt inputs[0]}, {NULL,NULL},
      {FloatIncDec,(caddr_t)&flt_inputs[1]}, {NULL,NULL},
      {FloatIncDec,(caddr_t)&flt_inputs[2]}, {NULL,NULL},
      {FloatIncDec,(caddr_t)&flt_inputs[3]}, {NULL,NULL},
      {FloatIncDec,(caddr_t)&flt_inputs[4]}, {NULL,NULL},
\}, vid call[]={
      {ChangeYN,(caddr_t)&ctrl-> stillvid}, {NULL,NULL},
      {FloatIncDec,(caddr_t)&flt_inputs[5]}, {NULL,NULL},
      {NumIncDec,(caddr t)&num_inputs[0]}, {NULL,NULL},
      {NumIncDec,(caddr_t)&num_inputs[0]}, {NULL,NULL},
      {ChangeYN,(caddr_t)&ctrl->auto_q}, {NULL,NULL},
      {ChangeYN,(caddr_t)&ctrl-> buf_switch}, {NULL,NULL},
      {NumIncDec,(caddr_t)&num_inputs[1]}, {NULL,NULL},
      {NumIncDec.(caddr_t)&num_inputs[1]}, {NULL, NULL},
\}, dec_call[] = {
      {SimpleMenu,(caddr_t)&ctrl->decide}, {NULL,NULL},
```

```
{SimpleMenu,(caddr_t)&ctrl->decide}, {NULL,NULL},
      {SimpleMenu,(caddr_t)&ctrl->decide}, {NULL,NULL},
}, filt call[]={
      {SimpleMenu,(caddr_t)&ctrl->filter}, {NULL,NULL},
      {SimpleMenu,(caddr_t)\&ctrl-> filter}, {NULL,NULL},
}, oct call[2*space];
XFontStruct *font;
Arg
      args[1];
msg->rows=1; msg->cols=NAME_LEN;
msg stats->rows=1; msg_stats->cols=NAME_LEN;
msg bin->rows=1; msg_bin->cols=NAME_LEN;
ctrl->src=(Video)closure;
flt_inputs[0].format="Quant: %4.1f";
flt inputs[0].max = 10;
flt inputs[0].min=0;
flt_inputs[0].value = &ctrl-> quant_const;
flt inputs[1].format="Thresh: %4.1f";
flt inputs[1].max = 10;
flt_inputs[1].min=0;
flt_inputs[1].value = &ctrl-> thresh_const;
flt_inputs[2].format="Comp: %4.1f";
flt inputs[2].max = 10;
flt inputs[2].min=0;
flt inputs[2].value = &ctrl-> cmp_const;
flt_inputs[3].format="Chrome: %4.1f";
 flt inputs[3].max = 5;
 flt inputs[3].min=1;
```

```
flt inputs[3].value = &ctrl->chrome_factor;
flt inputs[4].format="Diag: %4.1f";
fit inputs[4].max = 2.0;
flt inputs[4].min=1.0;
flt_inputs[4].value = &ctrl->diag_factor;
flt inputs[5].format="Target: %4.1f";
flt inputs[5].max = 30.0;
flt inputs[5].min=10.0;
flt inputs[5].value = &ctrl-> fps;
num inputs[0].format="px64k: %1d";
num inputs[0].max=8;
num inputs[0].min=1;
num_inputs[0] value = &ctrl-> bitrate;
num_inputs[1].format = "History: %1d";
num_inputs[1].max = 8;
num inputs[1].min=1;
num_inputs[1].value = &ctrl-> feedback;
for(i=0;i < space;i++)
       String format=(char *)MALLOC(20);
       if (i = 0) sprintf(format, "Octave LPF: %%4.2f");
       else sprintf(format, "Octave %3d: %%4.2f", space-i-1);
       oct_inputs[i].format = format;
       oct_inputs[i].max = 1.0;
       oct_inputs[i].min=0.0;
       oct_inputs[i].value = &ctrl->base_factors[space-i-1];
       oct items[2*i].name = "klics_oct_float";
```

```
oct items[2*i].contents=NULL;
             oct items[2*i].fromHoriz=0;
             oct_items[2*i].fromVert=i = 0.0:2*i-1;
             oct items[2*i].type=FW_float;
             oct items[2*i].hook=(String)&oct_inputs[i];
             oct_items[2*i+1].name="klics_oct_scroll";
             oct_items[2*i+1].contents=NULL;
             oct items[2*i+1].fromHoriz=1;
             oct_items[2*i+1].fromVert=i=0?0:2*i-1;
             oct items[2*i+1].type=FW_scroll;
             oct items[2*i+1].hook=(String)&oct_inputs[i];
             oct_call[2*i].callback=FloatIncDec;
             oct call[2*i].closure = (String)&oct_inputs[i];
             oct call[2*i+1].callback=NULL;
             oct cail[2*i+1].closure=NULL;
      }
      FillForm(form, COMP ICONS-(video-> size[2] > 1?0:1), items, widgets, callbacks);
      FillForm(widgets[23], 2*space, oct_items, oct_widgets, oct_call);
      FillMenu(dec shell, THREE, dec_menu, dec_widgets, dec_call);
      font=FindFont(widgets[12]);
XtSetArg(args[0],XtNwidth,2+TextWidth(0,"Maximum\nSigmaAbs\nSigmaSqr",font));
      XtSetValues(widgets[12], args, ONE);
      if (video-> size[2] > 1) {
             FillForm(widgets[24], VID ICONS, vid items, vid_widgets, vid_call);
             FillMenu(filt_shell,TWO,filt_menu,filt_widgets,filt_call);
             font=FindFont(vid widgets[11]);
             XtSetArg(args[0],XtNwidth,2+TextWidth(0,"None\nExp",font));
              XtSetValues(vid widgets[11], args, ONE);
       XtPopup(shell, XtGrabExclusive);
}
```

source/KlicsSA.c

/* Full still/video Knowles-Lewis Image Compression System utilising HVS properties and delta-tree coding Stand-Alone version uses fixed image format and static data structures */ #include "KlicsSA.h" #include < math.h > extern void Convolve(); /* useful X definitions */ typedef char Boolean; #define True 1 #define False 0 #define String char* /* token modes (empty) */ #define **EMPTY** #define CHANNEL EMPTY OCTAVE_EMPTY #define #define LPF_EMPTY #define FULL

- /* Function Name: AccessSA
- * Description: Find index address from co-ordinates
- * Arguments: x, y (x,y) co-ordinates

```
oct, sub, channel - octave, sub-band and channel co-ordinates
      Returns: index into data[channel][][index]
*/
      AccessSA(x,y,oct,sub,channel)
int
      x, y, oct, sub, channel;
int
{
return(((x < < 1) + (sub > > 1) + (SA_WIDTH > > (channel = = 0?0:1))*((y < < 1) + (1&sub))
)) < < oct);
}
      Function Name:
                           DecideSA
/*
      Description: Calculate value representing the difference between new and old
blocks
       Arguments: new, old - blocks to compare
                    difference value
       Returns:
 */
       DecideSA(new,old)
int
Block new, old;
{
              X, Y, sigma=0;
       int
       for(X=0;X < BLOCK;X++) for(Y=0;Y < BLOCK;Y++)
 sigma + = abs(new[X][Y]-old[X][Y]);
       return(sigma);
 }
```

```
/*
      Function Name:
                           DecideDoubleSA
      Description: Calculates normal w.r.t differencing algorithm
      Arguments:
                    norm - normal value
      Returns:
                    new normal value
 */
double
             DecideDoubleSA(norm)
double
             norm;
{
      return(4.0*norm);
Boolean
             DecisionSA(new,old,norm)
Block new, old;
double
             norm;
{
      return((double)DecideSA(new,old) < = DecideDoubleSA(norm));
}
                          HuffmanSA
/*
      Function Name:
      Description: Calculates the number of bits for the Huffman code representing
level
      Arguments:
                   level - level to be encoded
                    number of bits in codeword
      Returns:
 */
      HuffmanSA(level)
int
```

```
level;
int
{
       return(level = = 0?2:(abs(level) < 3?3:1 + abs(level)));
}
       Function Name:
                            HuffCodeSA
/*
       Description: Generates Huffman code representing level
                     level - level to be encoded
       Arguments:
                     coded bits in char's
       Returns:
 */
unsigned char *HuffCodeSA(level)
int
       level;
{
       unsigned char *bytes=(unsigned char *)MALLOC((7+Huffman(level))/8);
       bytes[0] = (abs(level) < 3?abs(level):3) | (level < 0?4:0);
       if (abs(level) > 2) {
                     index = (7 + Huffman(level))/8-1;
              int
              bytes[index] = bytes[index] | (1 < (Huffman(level)-1)\%8);
       }
       return(bytes);
}
unsigned char *CodeIntSA(number,bits)
       number, bits;
int
```

```
{
       int
              len = (7 + bits)/8;
       unsigned char *bytes=(unsigned char *)MALLOC(len);
       int
              byte;
       for(byte=0;byte<len;byte++) {
              bytes[byte] = 0xff&number;
              number = number > > 8;
       }
       return(bytes);
}
int
       ReadIntSA(bits,bfp)
int
       bits;
Bits
       bfp;
{
              len=(7+bits)/8;
       int
       unsigned char bytes[len];
              byte, number = 0;
       int
       bread(bytes,bits,bfp);
       for(byte=0;byte < len;byte++)
              number = number | ((int)bytes[byte] < < byte*8);
       number = (number < < sizeof(int)*8-bits) > > sizeof(int)*8-bits;
       return(number);
}
       Function Name:
                            HuffReadSA
        Description: Read Huffman encoded number from binary file
                     bfp - binary file pointer
        Arguments:
```

```
Returns:
                     decoded level
*/
int
       HuffReadSA(bfp)
Bits
       bfp;
{
              value;
       int
       unsigned char
                             byte;
       Boolean
                      negative = False;
       bread(&byte,2,bfp);
       value = (int)byte;
       if (byte = = '\0') return(0);
       else {
              bread(&byte,1,bfp);
              negative = (byte! = '\0');
       }
       if (value < 3) return(negif(negative, value));
       for(byte = '\0';byte = = '\0';value + +) bread(\&byte,1,bfp);
       return(negif(negative, value-1));
}
       Function Name:
                             QuantizeSA
/*
       Description: RM8 style quantizer
       Arguments:
                      data - unquantised number
                             q - quantizing divisor
                             level - quantised to level
                      quantized data & level
        Returns:
 */
```

```
QuantizeSA(data,q,level)
int
      data, q, *level;
int
{
      int
             mag level = abs(data)/q;
       *level = negif(data < 0, mag_level);
      return(negif(data < 0, mag_level*q+(mag_level! = 0?(q-1) > 1:0)));
}
/*
      Function Name:
                           ProposedSA
      Description: Calculates proposed block values
      Arguments: pro - proposed block
                           lev - proposed block quantized levels
                           old, new - old and new block values
                           norms - HVS normals
                    new = = 0, proposed values (pro) and levels (lev)
      Returns:
Boolean
             ProposedSA(pro,lev,old,new,norms)
Block pro, lev, old, new;
double
             norms[3];
{
      Block zero_block=\{\{0,0\},\{0,0\}\};
             X, Y, step = norms[0] < 1.0?1:(int)norms[0];
      int
                    zero = DecisionSA(new,zero_block,norms[1]);
      Boolean
      for(X=0;X < BLOCK;X++) for(Y=0;Y < BLOCK;Y++)
```

```
pro[X][Y] = zero?0:old[X][Y] + Quantize(new[X][Y]-old[X][Y], step, &(lev[X][Y]));
      return(zero);
}
                           ZeroCoeffsSA
      Function Name:
/*
      Description: Zero out video data
      Arguments: data - image data
                           addr - addresses
                    zeros data[addr[[]]]
       Returns:
 */
      ZeroCoeffsSA(data,addr)
void
short *data;
Block addr;
{
             X, Y;
       int
       for(X=0;X < BLOCK;X++) for(Y=0;Y < BLOCK;Y++)
              data[addr[X][Y]] = 0;
}
       Function Name:
                           BlockZeroSA
/*
       Description: Test if all block values are zero
       Arguments:
                    block - block under test
                    block = = 0
       Returns:
 */
              BlockZeroSA(block)
 Boolean
 Block block;
```

```
{
            X, Y;
      int
      Boolean
                  zero = True;
      for(X=0;X < BLOCK;X++) for(Y=0;Y < BLOCK;Y++)
            if (block[X][Y]!=0) zero=False;
      return(zero);
}
                         SendTokenSA
      Function Name:
      Description: Increments token frequency
      Arguments: token - token to be transmitted
                         channel, sub, oct - co-ordinates
                         bfp - binary file pointer
                         empty - zero state {EMPTY | CHANNEL_EMPTY |
OCTAVE_EMPTY | LPF_EMPTY | FULL}
                         branch - branch of tree (0-3)
                   encodes token
      Returns:
 */
      SendTokenSA(token,channel,sub,oct,bfp,empty,branch)
void
      token, channel, sub, oct, *empty, branch;
int
      bfp;
Bits
{
            full=FULL, i;
      int
      String
token_name[TOKENS] = {"ZERO_STILL","NON_ZERO_STILL","BLOCK_SAME","ZE
RO_VID","BLOCK_CHANGE",
 "LOCAL_ZERO", "LOCAL_NON_ZERO", "CHANNEL_ZERO", "CHANNEL_NON_ZE
```

```
RO". "OCT ZERO", "OCT NON_ZERO",
"LPF_ZERO"."LPF_NON_ZERO","LPF_LOC_ZERO"."LPF_LOC_NON_ZERO"};
                   switch(*empty) {
                   case EMPTY:
                                      if (token! = ZERO_STILL && token! = BLOCK_SAME) {
SendTokenSA(LOCAL\_NON\_ZERO, channel, sub, oct, bfp, \&full, branch);
                                                          for(i=0;i < channel;i++)
 SendTokenSA(CHANNEL\_ZERO, i, sub, oct, bfp, \& full, branch);
                                                          *empty=CHANNEL_EMPTY;
                                                          SendTokenSA (token, channel, sub, oct, bfp, empty, branch);\\
                                        }
                                        break:
                     case CHANNEL EMPTY:
                                        if (token! = ZERO_STILL && token! = BLOCK_SAME) {
  SendTokenSA(CHANNEL_NON_ZERO,channel,sub,oct,bfp,&full,branch);
                                                          for(i=1;i < sub;i++)
   SendTokenSA(token = = NON\_ZERO\_STILL?ZERO\_STILL:BLOCK\_SAME, channel, i, occording to the control of the contr
   t,bfp,&full,branch);
                                                            *empty=FULL;
                                                           SendTokenSA(token,channel,sub,oct,bfp,empty,branch);
                                         }
                                         break;
                      case OCTAVE_EMPTY:
                                         if (token! = ZERO_STILL && token! = BLOCK_SAME) {
    SendTokenSA(OCT\_NON\_ZERO, channel, sub, oct, bfp, \&full, branch);
                                                            for(i=0; i < branch; i++)
     SendTokenSA(token = = NON_ZERO_STILL?ZERO_STILL:BLOCK_SAME,channel,sub
```

```
.oct,bfp,&full,branch);
                    *empty = FULL;
                    SendTokenSA(token,channel,sub,oct,bfp,empty,branch);
             break;
      case LPF_EMPTY:
             if (token!=LPF_ZERO) {
SendTokenSA(LPF LOC_NON_ZERO,channel,sub,oct,bfp,&full,branch);
                    for(i=0; i < channel; i++)
SendTokenSA(LPF_ZERO,i,sub,oct,bfp,&full,branch);
                    *empty = FULL;
                    SendTokenSA(token,channel,sub,oct,bfp,empty,branch);
             }
             break:
       case FULL:
             Dprintf("%s\n",token_name[token]);
             bwrite(&token_codes[token],token_bits[token],bfp);
             break;
       }
}
                           ReadBlockSA
       Function Name:
       Description: Read block from video
       Arguments: new, old, addr - new and old blocks and addresses
                           x, y, oct, sub, channel - co-ordinates of block
                           src, dst - frame data
       Returns:
                    block values
 */
       ReadBlockSA(new.old,addr,x,y,oct,sub,channel,src,dst)
void
```

```
Block new, old, addr;
      x, y, oct, sub, channel;
int
short *src[3], *dst[3];
{
             X, Y;
      int
      for(X=0;X < BLOCK;X++) for(Y=0;Y < BLOCK;Y++) 
              addr[X][Y] = AccessSA((x < < 1) + X,(y < < 1) + Y,oct,sub,channel);
             new[X][Y] = (int)src[channel][addr[X][Y]];
              old[X][Y] = (int)dst[channel][addr[X][Y]];
       }
}
                           CalcNormalsSA
       Function Name:
/*
       Description: Calculates HVS weighted normals
       Arguments: oct, sub, channel - co-ordinates
                           norms - pre-initialised normals
                     weighted normals
       Returns:
 */
       CalcNormals SA(oct, sub, channel, norms, quant\_const)
void
       oct, sub, channel;
 int
              norms[3], quant_const;
 double
 {
              norm, base_oct=oct+(channel!=0?1:0)+(sub==0?1:0);
        int
        for(norm=0;norm<3;norm++) {
              if (norm!=0) norms[norm] *= quant_const;
              norms[norm] *= base_factors[base_oct]*(sub = = 3?diag_factor: 1.0);
```

```
if (channel! = 0) norms[norm] *= chrome_factor;
            norms[norm] *=(double)(1 < SA PRECISION);
      }
}
      Function Name:
                        MakeDecisions2SA
      Description: Decide on new compression mode from block values
                   old, new, pro - block values
      Arguments:
                          zero - zero flag for new block
                          norms - HVS normals
                          mode - current compression mode
                          decide - comparison algorithm
                   new compression mode
      Returns:
      MakeDecisions2SA(old,new,pro,lev,zero,norms,mode)
int
Block new, old, pro, lev;
Boolean
             zero:
             norms[3];
double
int
      mode;
{
      Block zero_block = \{\{0,0\},\{0,0\}\}\;
             new_mode=mode==STILL || BlockZeroSA(old)?STILL:SEND,
      int
                   np=DecideSA(new,pro), no=DecideSA(new,old);
      if (new_mode = = STILL) new_mode = np > = no | | zero | |
BlockZeroSA(lev)?STOP:STILL;
      else new mode=zero && np<no?VOID:np>=no | |
DecisionSA(new,old,norms[2]) | | BlockZeroSA(lev)?STOP:SEND;
       return(new mode);
```

```
}
       Function Name:
                           UpdateCoeffsSA
/*
       Description: Encode proposed values and write data
                    pro, lev, addr - proposed block, levels and addresses
       Arguments:
                           channel, oct - co-ordinates
                           dst - destination data
                           bfp - binary file pointer
                    alters dst[channel][addr[][]]
       Returns:
 */
      UpdateCoeffsSA(pro,lev,addr,channel,oct,dst,bfp)
void
Block pro, lev, addr;
       channel, oct;
int
       *dst[3];
short
Bits
       bfp;
{
             X, Y;
       int
       for(X=0;X < BLOCK;X++) for(Y=0;Y < BLOCK;Y++) 
                     bits=HuffmanSA(lev[X][Y]),
              int
                           level = abs(lev[X][Y]);
                                   *bytes=HuffCodeSA(lev[X][Y]);
              unsigned char
              dst[channel][addr[X][Y]] = (short)pro[X][Y];
              bwrite(bytes,bits,bfp);
              XtFree(bytes);
       }
}
```

```
SendTreeSA
      Function Name:
      Description: Encode tree blocks
                    prev_mode - compression mode
      Arguments:
                           x, y, oct, sub, channel - co-ordinates
                           empty - token mode
                           branch - tree branch number
                    active block indicator
       Returns:
*/
Boolean
SendTreeSA(prev_mode.x,y,oct,sub,channel,src,dst,empty,branch,quant_const,bfp)
       prev_mode, x, y, oct, sub, channel, *empty, branch;
int
short *src[3], *dst[3];
              quant const;
double
Bits
       bfp;
{
       Block addr. old, new, pro, lev;
              new mode, X, Y;
       int
                     norms[3] = {quant_const.thresh_const.cmp_const}; /* quant. thresh.
       double
compare */
                     active = False:
       Boolean
       ReadBlockSA(new.old.addr.x.y,oct.sub.channel.src.dst);
       if (prev mode! = VOID) {
              Boolean
                            zero:
              CalcNormalsSA(oct.sub.channel.norms.quant_const);
              zero = ProposedSA(pro, lev, old, new, norms);
              new_mode = MakeDecisions2SA(old.new.pro.lev.zero.norms.prev_mode);
              switch(new_mode) {
```

```
case STOP:
                                                                   SendTokenSA(prev mode = = STILL | |
BlockZeroSA(old)?ZERO_STILL:BLOCK_SAME.channel.sub,oct.bfp,empty,branch);
                                                                   break:
                                            case STILL:
                                            case SEND:
                                                                   active = True:
                                                                   SendTokenSA(prev_mode = = STILL | |
BlockZero(old)?NON_ZERO_STILL:BLOCK_CHANGE.channel.sub.oct.bfp,empty,bran
ch);
                                                                    UpdateCoeffsSA(pro,lev,addr,channel,oct,dst,bfp);
                                                                    break:
                                             case VOID:
                                                                     SendTokenSA(ZERO_VID,channel,sub,oct,bfp,empty,branch);
                                                                     ZeroCoeffsSA(dst[channel],addr);
                                                                    break:
                        } else {
                                              if (BlockZeroSA(old)) new_mode=STOP;
                                              eise {
                                                                     ZeroCoeffsSA(dst[channel],addr);
                                                                     new_mode = VOID;
                                               }
                         if (oct>0 && new_mode!=STOP) {
                                                                     mt=OCTAVE_EMPTY, full=FULL;
                                               int
                                               Dprintf("x = \%d, y = \%d, oct = \%d sub = \%d mode
    %d\n".x,y.oct.sub.new_mode);
                                               for(Y=0;Y<2;Y++) for(X=0;X<2;X++)
    (void) Send Tree SA (new\_mode.x + 2 + X.y + 2 + Y.oct - 1.sub. channel.src.dst. \&mt. X + 2 + Y.qua) Send Tree SA (new\_mode.x + 2 + X.y + 2 + Y.oct - 1.sub. channel.src.dst. \&mt. X + 2 + Y.qua) Send Tree SA (new\_mode.x + 2 + X.y + 2 + Y.oct - 1.sub. channel.src.dst. \&mt. X + 2 + Y.qua) Send Tree SA (new\_mode.x + 2 + X.y + 2 + Y.oct - 1.sub. channel.src.dst. \&mt. X + 2 + Y.qua) Send Tree SA (new\_mode.x + 2 + X.y + 2 + Y.oct - 1.sub. channel.src.dst. \&mt. X + 2 + Y.qua) Send Tree SA (new\_mode.x + 2 + X.y + 2 + Y.oct - 1.sub. channel.src.dst. \&mt. X + 2 + Y.qua) Send Tree SA (new\_mode.x + 2 + X.y + 2 + Y.oct - 1.sub. channel.src.dst. \&mt. X + 2 + Y.qua) Send Tree SA (new\_mode.x + 2 + X.y + 2 + Y.oct - 1.sub. channel.src.dst. \&mt. X + 2 + Y.qua) Send Tree SA (new\_mode.x + 2 + X.y + 2 + Y.oct - 1.sub. channel.src.dst. \&mt. X + 2 + Y.qua) Send Tree SA (new\_mode.x + 2 + X.y + 2 + Y.oct - 1.sub. channel.src.dst. \&mt. X + 2 + Y.qua) Send Tree SA (new\_mode.x + 2 + X.y + 2 + Y.oct - 1.sub. channel.src.dst. \&mt. X + 2 + Y.qua) Send Tree SA (new\_mode.x + 2 + X.y + 2 + Y.qua) Send Tree SA (new\_mode.x + 2 + X.y + 2 + Y.qua) Send Tree SA (new\_mode.x + 2 + X.y + 2 + Y.qua) Send Tree SA (new\_mode.x + 2 + X.y + 2 + Y.qua) Send Tree SA (new\_mode.x + 2 + X.y + 2 + Y.qua) Send Tree SA (new\_mode.x + 2 + X.y + 2 + Y.qua) Send Tree SA (new\_mode.x + 2 + X.y + 2 + Y.qua) Send Tree SA (new\_mode.x + 2 + X.y + 2 + Y.qua) Send Tree SA (new\_mode.x + 2 + X.y + 2 + Y.qua) Send Tree SA (new\_mode.x + 2 + X.y + 2 + Y.qua) Send Tree SA (new\_mode.x + 2 + X.y + 2 + Y.qua) Send Tree SA (new\_mode.x + 2 + X.y + 2 + Y.qua) Send Tree SA (new\_mode.x + 2 + X.y + 2 + Y.qua) Send Tree SA (new\_mode.x + 2 + X.y + 2 + Y.qua) Send Tree SA (new\_mode.x + 2 + X.y + 2 + Y.qua) Send Tree SA (new\_mode.x + 2 + X.y + 2 + Y.qua) Send Tree SA (new\_mode.x + 2 + X.y + 2 + Y.qua) Send Tree SA (new\_mode.x + 2 + X.y + 2 + Y.qua) Send Tree SA (new\_mode.x + 2 + X.y + X.y
```

```
nt const.bfp);
             if (mt = = OCTAVE EMPTY && new_mode! = VOID)
SendTokenSA(OCT ZERO, channel. sub.oct. bfp, & full. 0);
      }
      return(active);
}
      Function Name:
                           SendLPF_SA
      Description: Encode LPF sub-band
       Arguments:
                    mode - compression mode
                    encodes data
       Returns:
 */
       SendLPF_SA(mode,src,dst,bfp,quant_const)
void
int
       mode;
      *src[3], *dst[3];
short
Bits bfp;
double
             quant_const;
{
       Block new, old, pro, lev, addr;
             channel, channels=3, x, y, full=FULL.
       int
                    octs_lum = 3,
size[2] = {SA WIDTH > > octs_lum + 1.SA_HEIGHT > > octs_lum + 1};
       for(y = 0:y < size[1];y + +) for(x = 0;x < size[0];x + +) {
                    empty=LPF EMPTY;
             int
       for(channel = 0:channel < channels:channel + +) {
                    octs = channel! = 0.2:3.
              int
```

```
new mode, X, Y, step, value, bits=0;
             double
                          norms[3] = {quant const.thresh const.cmp const};
             CalcNormalsSA(octs-1,0,channel.norms.quant const);
             step = norms[0] < 1.0?1:(int)norms[0];
             for(bits=0, value=((1 < 8 + SA_PRECISION)-1)/step; value!=0; bits++)
                   value = value > > 1;
             ReadBlockSA(new,old,addr,x,y,octs-1,0,channel,src,dst);
             /* Proposed */
             for(X=0;X < BLOCK;X++) for(Y=0;Y < BLOCK;Y++)
pro[X][Y] = old[X][Y] + QuantizeSA(new[X][Y] - old[X][Y], step, &(lev[X][Y]));
             /* MakeDecisions */
             new mode=mode==STILL?STILL:DecisionSA(new,old,norms[2]) | |
BlockZeroSA(lev)?STOP:SEND;
             switch(new_mode) {
             case SEND:
                   SendTokenSA(LPF NON ZERO, channel, 0, octs, bfp, &empty, 0);
                   UpdateCoeffsSA(pro,lev,addr,channel,octs,dst,bfp);
             break:
             case STILL:
                   for(X=0;X < BLOCK;X++) for(Y=0;Y < BLOCK;Y++) 
                          unsigned char *bytes=CodeIntSA(lev[X][Y],bits);
                          dst[channel][addr[X][Y]] = (short)pro[X][Y];
                          bwrite(bytes.bits.bfp);
                          XtFree(bytes);
                   break:
```

```
case STOP:
                    SendTokenSA(LPF_ZERO.channel.0.octs.bfp.&empty,0);
                    break:
              }
       if (mode! = STILL && empty = = LPF EMPTY)
SendTokenSA(LPF LOC ZERO, channel. 0, octs_lum.bfp, &full.0);
}
       Function Name:
                           CompressFrameSA
       Description: Compress a Frame
                    mode - compression mode STILL or SEND
       Arguments:
                           src, dst - source and destination data
                           bfp - binary file pointer for result
                           quant_const - quantization parameter
void
      CompressFrameSA(mode, src, dst, bfp, quant const)
int
      mode:
short
      *src[3], *dst[3];
Bits
      bfp;
double
             quant_const;
             sub, channel, x, y, i,
      int
                    octs_lum = 3,
size[2] = \{SA \ WIDTH > > 1 + octs_lum.SA_HEIGHT > > 1 + octs_lum\};
      for(channel = 0:channel < 3:channel + +) {
```

```
int
frame size[2] = \{SA\_WIDTH > > (channel = = 0?0:1).SA\_HEIGHT > > (channel = = 0?0:1).
)}.
                           frame area = frame_size[0]*frame size[1];
             for(i=0;i < frame area;i++)
src[channel][i] = src[channel][i] < < SA_PRECISION;</pre>
             Convolve(src[channel], False, frame_size, 0, channel = = 0?3:2);
       bwrite((char *)&quant const.sizeof(double)*8,bfp);
       SendLPF SA(mode, src, dst, bfp, quant_const);
       for(y=0;y < size[1];y++) for(x=0;x < size[0];x++) {
                    empty = EMPTY, full = FULL;
              int
              for(channel=0;channel<3;channel++) {
                           octs = channel! = 0?2:3;
                     int
                    for(sub=1;sub<4;sub++)
(void)SendTreeSA(mode,x,y,octs-1,sub,channel,src,dst,&empty,0,quant_const,bfp);
                    switch(empty) {
                    case FULL:
                           empty = CHANNEL_EMPTY;
                           break:
                    case CHANNEL_EMPTY:
SendTokenSA(CHANNEL ZERO.channel.sub,octs-1,bfp,&full,0);
                           break:
                     }
              if (empty = = EMPTY)
SendTokenSA(LOCAL ZERO, channel, sub, octs_lum-1, bfp, &full, 0);
       }
```

source/KlicsTestSA.c

```
#include
             "xwave.h"
#include
             "KlicsSA.h"
extern void CompressFrameSA();
             struct {
typedef
       Video src:
      char bin_name[STRLEN];
      Boolean
                    stillvid;
      double
                    quant const;
} KlicsCtrlRec, *KlicsCtrl;
      Function Name:
                           KlicsCtrlSA
/*
      Description: Test harness for KlicsSA in xwave
                           w - Xaw widget
       Arguments:
                                  closure - compression control record
                                 call_data - NULL
                  send data to binary file
      Returns:
 */
     KlicsCtrlSA(w,closure,call_data)
void
Widget
             closure, call_data;
caddr t
{
                    ctrl=(KlicsCtrl)closure:
      KlicsCtrl
             sizeY = SA WIDTH*SA HEIGHT,
      int
```

```
sizeUV=SA_WIDTH*SA_HEIGHT/4, i, z;
      short *dst[3]={}
             (short *)MALLOC(sizeof(short)*sizeY),
             (short *)MALLOC(sizeof(short)*sizeUV),
             (short *)MALLOC(sizeof(short)*sizeUV),
      (short *)MALLOC(sizeof(short)*sizeY),
             (short *)MALLOC(sizeof(short)*sizeUV),
             (short *)MALLOC(sizeof(short)*sizeUV),
      };
      char
             file_name(STRLEN);
      Bits
             bfp;
                   true=True, false=False;
      Boolean
      for(i=0; i < size Y; i++) dst[0][i]=0;
      for(i=0;i < sizeUV;i++) \{ dst[1][i]=0; dst[2][i]=0; \}
sprintf(file name, "%s%s/%s%s\0", global->home.KLICS_SA_DIR,ctrl->bin name.KLI
CS_SA_EXT);
      bfp=bopen(file name, "w");
      bwrite(&ctrl-> stillvid, 1, bfp);
      bwrite(&ctrl-> src-> size[2], sizeof(int) *8, bfp);
      for(z=0;z<ctrl->src->size[2];z++)
             GetFrame(ctrl-> src,z);
             for(i=0;i < sizeY;i++) src[0][i] = ctrl-> src-> data[0][z][i];
             for(i=0;i < sizeUV;i++)
                   src[1][i] = ctrl-> src-> data[1][z][i];
                   src[2][i] = ctrl -> src -> data[2][z][i];
             CompressFrameSA(z = 0 | |
```

```
ctrl->stillvid?STILL:SEND.src,dst.bfp,ctrl->quant_const);
               FreeFrame(ctrl->src.z);
        bflush(bfp);
        bclose(bfp);
        XtFree(dst[0]);
        XtFree(dst[1]);
        XtFree(dst[2]);
        XtFree(src[0]);
        XtFree(src[1]);
        XtFree(src[2]);
 }
 KlicsCtrl
               InitKlicsCtrl(name)
 String name;
 {
        KlicsCtrl
                     ctrl=(KlicsCtrl)MALLOC(sizeof(KlicsCtrlRec));
        ctrl-> stillvid = True;
       ctri->quant_const = 8.0;
        strcpy(ctrl->bin_name.name);
        return(ctri);
 #define
               KLICS_SA_ICONS 8
 #define KLICS_SA_VID_ICONS
 void
        KlicsSA(w.closure.call_data)
Widget
               w;
```

```
closure, call data;
caddr t
{
      Video video = (Video) closure:
      KlicsCtrl
                    ctrl=InitKlicsCtrl(video-> name);
      FloatInput
                    flt_inputs = (FloatInput)MALLOC(sizeof(FloatInputRec));
                    msg_bin=NewMessage(ctrl->bin_name,NAME LEN);
      Message
      XtCallbackRec
                           destroy call[]={
             {Free.(caddr t)ctrl}.
             {Free,(caddr_t)flt_inputs},
             {CloseMessage,(caddr_t)msg_bin},
             {NULL, NULL},
      };
      Widget
                    parent = FindWidget("frm_compress", XtParent(w)),
shell=ShellWidget("klicsSA", parent,SW_below, NULL, destroy call),
                    form = Format Widget("klicsSA form", shell),
                    widgets[KLICS_SA_ICONS],
vid_widgets[KLICS_SA_VID_ICONS];
                    items() = {
      Formitem
             {"klicsSA cancel", "cancel", 0, 0, FW icon, NULL}.
             {"klicsSA_confirm", "confirm", 1,0,FW_icon, NULL}.
             {"klicsSA_title", "Run Klics SA", 2, 0, FW_label, NULL},
             {"klicsSA_bin_lab", "KLICS File:",0,3,FW label,NULL}.
             {"klicsSA_bin_name", NULL, 4, 3, FW text, (String) msg_bin},
             {"klicsSA_qn_float".NULL.0,5,FW_float,(String)&flt_inputs[0]},
             {"klicsSA_qn_scroll", NULL.6,5,FW_scroll.(String)&flt_inputs[0]},
             {"klicsSA vid form".NULL.0,7,FW form.NULL},
      \}, vid items()={
             {"klicsSA ic_lab", "Image Comp: ",0,0,FW label.NULL},
             {"klicsSA_ic".NULL.1.0.FW_yn,(String)&ctrl->stillvid},
      };
```

}

```
XtCallbackRec
                          callbacks[]={
            {Destroy,(caddr_t)shell},
            {NULL.NULL},
            {KlicsCtrlSA.(caddr_t)ctrl}.
            {Destroy,(caddr_t)shell},
            {NULL, NULL},
            {FloatIncDec.(caddr_t)&flt_inputs[0]}, {NULL,NULL}.
      }, vid_call[]={
            {ChangeYN,(caddr t)&ctrl-> stillvid}, {NULL, NULL},
      };
      ctrl-> src = video:
      msg_bin->rows=1; msg_bin->cols=NAME_LEN;
      flt inputs[0].format="Quant: %4.1f";
      flt inputs[0].max=10;
      flt inputs[0].min=0;
      flt inputs[0].value = &ctrl->quant_const;
FillForm(form.KLICS_SA_ICONS-(video->size[2]>1?0:1), items, widgets, callbacks);
      if (video - size[2] > 1)
FillForm(widgets[7],KLICS_SA_VID_ICONS,vid_items,vid_widgets,vid_call);
      XtPopup(shell,XtGrabExclusive);
```

source/Malloc.c

source/Menu.c

```
Pull-Right Menu functions
       */
#include
              < stdio.h >
#include
              <X11/IntrinsicP.h>
             <X11/StringDefs.h>
#include
#include -
             <X11/Xaw/XawInit.h>
#include
             < X11/Xaw/SimpleMenP.h>
             <X11/Xaw/CommandP.h>
#include
static void
             prPopupMenu();
static void
             NotifyImage();
static void
             PrLeave();
void
      InitActions(app_con)
XtAppContext
                   app_con;
{
      static XtActionsRec
                                actions() = {
             {"prPopupMenu",prPopupMenu},
             {"notifyImage", NotifyImage},
             {"prLeave", PrLeave},
      };
      XtAppAddActions(app_con.actions.XtNumber(actions));
```

```
static void prPopupMenu(w.event.params.num_params)
Widget w:
XEvent * event;
String * params;
Cardinal * num_params;
 Widget menu. temp;
 Arg arglist[2];
 Cardinal num_args;
 int menu_x, menu_y, menu_width, menu_height, button_width, button_height;
 Position button_x, button_y;
 if (*num params!=1) {
      char error_buf[BUFSIZ];
      sprintf(error_buf, "prPopupMenu: %s.", "Illegal number of translation
arguments");
      XtAppWarning(XtWidgetToApplicationContext(w), error_buf);
      return:
 temp = w;
 while(temp != NULL) {
  menu = XtNameToWidget(temp, params[0]);
  if (menu == NULL)
    temp = XtParent(temp);
  else
    break:
```

```
if (memu = = NULL) {
 char error buf[BUFSIZ];
 sprintf(error_buf, "prPopupMenu: %s %s.".
        "Could not find menu widget named", params[0]);
 XtAppWarning(XtWidgetToApplicationContext(w), error_buf);
 return;
if (!XtIsRealized(menu))
 XtRealizeWidget(menu);
menu width = menu->core.width + 2 * menu->core.border_width;
button_width = w->core.width + 2 * w->core.border_width;
button height = w->core.height + 2 * w->core.border_width;
menu_height = menu->core.height + 2 * menu->core.border_width;
XtTranslateCoords(w, 0, 0, &button_x, &button_y);
menu_x = button_x;
menu y = button_y + button_height;
if (menu x < 0)
 menu x = 0;
else {
 int scr width = WidthOfScreen(XtScreen(menu));
 if (menu_x + menu_width > scr_width)
   menu_x = scr_width - menu_width:
if (menu y < 0)
 menu y = 0;
else {
 int scr height = HeightOfScreen(XtScreen(menu));
```

```
if (menu_y + menu_height > scr_height)
    menu y = scr_height - menu_height:
 }
 num args = 0:
 XtSetArg(arglist[num_args], XtNx, menu_x); num_args++;
 XtSetArg(arglist[num_args], XtNy, menu_y); num_args++;
 XtSetValues(menu. arglist, num_args);
 XtPopupSpringLoaded(menu);
}
static void
prRealize(w, mask, attrs)
Widget w;
Mask *mask:
XSetWindowAttributes *attrs;
 (*superclass->core_class.realize) (w, mask, aurs);
 /* We have a window now. Register a grab. */
 XGrabButton( XtDisplay(w), AnyButton, AnyModifier, XtWindow(w),
            TRUE, ButtonPressMask | ButtonReleaseMask,
            GrabModeAsync, GrabModeAsync, None, None);
}
*/
static void NotifyImage(w,event.params.num_params)
 Widget
             w;
 XEvent
              *event:
```

```
String *params:
Cardinal
             *num params:
      CommandWidget
                         cbw=(CommandWidget)w;
      if (cbw->command.set) XtCallCallbackList(w,cbw->command.callbacks,event);
}
static void PrLeave(w, event, params, num_params)
Widget
             w;
XEvent
             *event:
String *params;
Cardinal
             *num_params;
{
      SimpleMenuWidget smw=(SimpleMenuWidget)w;
      Dprintf("PrLeave\n");
}
```

WO 94/23385

- 257 -

source/Message.c

```
Message I/O Utility Routines
             "../include/xwave.h"
#include
#include
              < varargs.h>
             MESS_ICONS
                                  3
#define
     TextSize(msg)
void
Message
{
             i=-1, max_len=0;
      int
             *text=msg-> info.ptr;
      char
      msg-> rows=0;
      msg->cols=0;
      do {
             i++;
             if (text[i] = = '\n' \mid | text[i] = = '\0') 
                    if (msg->cols>max_len) max_len=msg->cols;
                    msg->cols=0;
                    msg-> rows++;
             } else msg->cols++;
      } while (text[i]! = '\0');
      if (i>0) if (text[i-1] = = 'n') msg-> rows--;
```

```
msg->cois=max_len;
}
             NewMessage(text.size)
Message
char
      *text;
int
      size;
{
                   msg = (Message)MALLOC(sizeof(MessageRec));
      Message
      msg-> shell=NULL;
      msg-> widget=NULL;
      msg->info.firstPos=0;
      if (!(msg->own_text=text==NULL)) msg->info.ptr=text;
      cise {
             msg->info.ptr=(char *)MALLOC(size+1);
             msg-> info.ptr[0] = '\0';
      msg->info.format=FMT8BIT;
      msg->info.length=0;
      msg->rows=0;
      msg->cois=0;
      msg->size=size;
      msg->edit=XawtextEdit;
      return(msg);
}
      CloseMessage(w,closure,call_data)
void
Widget
caddr t
             closure. call_data:
```

```
{
                    msg = (Message)closure;
      Message
       Destroy(w,(caddr_t)msg-> shell.NULL);
      if (msg->own_text) XtFree(msg->info.ptr);
      XtFree(msg);
}
      MessageWindow(parent,msg,title,close,call)
void
Widget
             parent;
Message
             msg;
       *title;
char
             close;
Boolean
XtCallbackRec
                    call[];
{
                    form, widgets[MESS_ICONS] = {NULL, NULL, NULL};
       Widget
                    items[] = {
       Formitem
             {"msg_cancel", "cancel", 0,0,FW_icon, NULL},
             {"msg_label",title,1,0,FW_label,NULL},
             {"msg_msg",NULL.0.2.FW_text,(String)msg},
       };.
       msg->edit=XawtextRead;
msg-> shell=ShellWidget("msg",parent,parent==global->toplevel?SW_top:SW_below,
NULL, NULL);
       form = FormatWidget("msg_form", msg-> shell);
FillForm(torm, MESS\_ICONS-(close?0:1), \& items[close?0:1], \& widgets[close?0:1], call);\\
       XtPopup(msg-> shell.XtGrabNone);
```

```
Mflush(msg);
}
      Mflush(msg)
void
Message
              msg;
{
       if (global->batch==NULL && msg->widget!=NULL) {
                           *dpy = XtDisplay(global-> toplevel);
              Display
              int
                     i, lines = 0;
                    args[1];
              Arg
              for(i=msg-> info.length-1:lines < msg-> rows && i>=0;i--)
                     if (msg-> info.ptr[i] = = '\n' && i! = msg-> info.length-1) lines + +;
              i++;
              if (msg-> info.ptr[i] = = '\n') i++;
              strcpy(msg-> info.ptr,&msg-> info.ptr[i]);
              msg->info.length-=i;
              XtSetArg(args[0], XtNstring, msg-> info.ptr);
              XSynchronize(dpy,True);
              XtSetValues(msg-> widget,args,ONE);
              XSynchronize(dpy,False);
       }
}
 void
       mprintf(msg,ap)
 Message
              msg;
 va_list
              ap;
```

```
*format;
      char
      format = va_arg(ap,char *);
      if (global->batch!=NULL) vprintf(format.ap);
      else {
                    text[STRLEN];
             char
              int
              vsprintf(text,format,ap);
             i=strien(text)+msg->info.length-msg->size;
              if (i>0) {
                     strcpy(msg->info.ptr,&msg->info.ptr[i]);
                     msg->info.length-=i;
              }
              strcat(msg-> info.ptr,text);
              msg->info.length+=strlen(text);
       }
}
void Dprintf(va_alist)
va_dcl
{
       va_list
                     ap;
       if (global->debug) {
              char
                     *format;
              va_start(ap);
              format = va_arg(ap,char *);
              vprintf(format.ap);
```

```
- 262 -
```

```
va_end(ap);
}
       Mprintf(va_alist)
void
va_dcl
{
       va_list
                     ap;
       Message
                     msg;
       va_start(ap);
       msg = va_arg(ap, Message);
       mprintf(msg,ap);
       va_end(ap);
}
       Eprintf(va_alist)
void
va_dcl
{
       va_list
                     ap;
       Message
              rows, cols;
       int
       va_start(ap);
       msg = NewMessage(NULL.STRLEN):
       mprintf(msg,ap);
       if (global->batch = = NULL) {
              XıCallbackRec
                                   callbacks[] = {
```

```
- 263 -
```

```
{CloseMessage,(caddr_t)msg},
{NULL,NULL},
};

TextSize(msg);
MessageWindow(global->toplevel,msg,"Xwave Error",True.callbacks);
}

va_end(ap);
```

source/NameButton.c

```
/*
      Supply MenuButton widget id to PullRightMenu button resource
*/
             "../include/xwave.h"
#include
     NameButton(w, event, params, num_params)
Widget
             W;
XEvent
             *event:
String *params;
Cardinal
             *num params;
{
      MenuButtonWidget mbw=(MenuButtonWidget) w;
      Widget
                   menu:
      Arg
             args[1];
      String name;
      XtSetArg(args[0], XtNmenuName, &name);
      XtGetValues(w, args, ONE);
      Dprintf("NameButton: looking for PRM %s\n",name);
      menu=FindWidget(name,w);
      if (menu!= NULL) {
                   Dprintf("NameButton: setting Menu Button\n");
                   XtSetArg(args[0], XtNbutton.w);
                   XtSetValues(menu.args,ONE);
      }
```

source/Palette.c

```
Palette re-mapping
*/
              "../include/xwave.h"
#include
/*
       Function Name:
                            ReMap
       Description: Re-maps a pixel value to a new value via a mapping
       Arguments: pixel - pixel value (0..max-1)
                            max - range of pixel values
                            map - palette to recode with
                     remapped pixel value
       Returns:
 */
       ReMap(pixel.max,palette)
int
int
       pixel. max;
Palette
              palette;
{
       Map map = palette- > mappings;
       int
              value = pixel;
                     inrange = False;
       Boolean
       while(map!=NULL && !inrange) {
              if (pixel > = map- > start && pixel < = map- > finish) {
                     inrange=True;
                     value = map -> m + pixel + map -> c;
```

```
- 266 -
```

```
map=map->next;
       }
       return(value < 0?0: value > = max?max-1: value);
}
/*
       Function Name:
                             FindPalette
       Description: Find a palette from a list given the index
       Arguments:
                      palette - the palette list
                             index - the index number
       Returns:
                      the palette corresponding to the index
 */
Palette
              FindPalette(palette, index)
Palette
              palette;
int
       index:
{
       while(index > 0 && palette-> next! = NULL) {
              index--;
              palette = palette- > next;
       }
       return(palette);
}
       Function Name:
/*
                            ReOrderPalettes
       Description: Reverse the order of the palette list
                     start, finish - the start and finish of the re-ordered list
       Returns: the palette list in the reverse order
*/
```

```
Palette ReOrderPalettes(start.finish)

Palette start, finish;

{

Palette list=finish->next;

if (list!=NULL) {

finish->next=list->next;

list->next=start;

start=ReOrderPalettes(list.finish);
}

return(start);
```

source/Parse.c

```
/*
       Parser for xwave input files: .elo
 */
#include
               "../include/xwave.h"
#include
               "../include/Gram.h"
       Parse(path.file.ext)
String path, file, ext;
{
             file_name[STRLEN];
      char
      sprintf(file_name, "%s%s/%s%s\0", global->home.path.file.ext);
      Dprintf("Parse: parsing file %s\n".file_name);
      if (NULL = =(global-> parse_fp=fopen(file_name, "r")))
             Eprintf("Parse: failed to open input file %s\n",file_name);
      else {
             sprintf(file_name, "%s%s\0", file.ext);
             global->parse_file=file_name:
             global->parse_token=ext;
            yyparse();
            fclose(global->parse_fp);
            Dprintf("Parse: finished with %s\n".file_name);
     }
```

```
ParseCtri(w,ciosure,call data)
void
Widget
              closure, call_data;
caddr_t
{
       Parse(".",((XawListReturnStruct *)call_data)-> string,(String)closure);
}
       Parseinput(fp)
int
FILE *fp;
{
       int
              num;
      if (global->parse_token! = NULL)
              if (global - parse\_token[0] = = '\0') {
                    num = (int)' \ ';
                    global->parse_token=NULL:
             } else {
                    num = (int)global-> parse_token[0];
                    giobal->parse_token++;
      else if (EOF = = (num = getc(global-> parse_fp))) num = NULL;
      return(num);
```

```
source/Pop2.c
```

```
/*
       Global callbacks for popping popups and allsorted utilities
*/
#include
              "../include/xwave.h"
      Destroy(w,closure,call_data)
void
Widget
caddr_t
             closure, call_data;
{
                    widget = (Widget) closure;
     Widget
       if (widget! = NULL) XtDestroyWidget(widget);
}
      Quit(w,closure,call_data)
void
Widget
              w;
caddr_t
             closure, call_data;
{
      XtDestroyApplicationContext(global->app_con);
      exit();
}
      Free (w.closure.call_data)
void
```

```
Widget
              w;
caddr_t
              closure, call data;
{
       if (closure! = NULL) XtFree(closure);
}
              FindWidget(name,current)
Widget
String name;
Widget
              current;
{
       Widget
                     target=NULL;
       while(current!=NULL) {
              target = XtNameToWidget(current,name);
              if (target = = NULL) current = XtParent(current);
              else break;
       if (target = = NULL) {
              Eprintf("Cant find widget: %s\n",name);
             target = global- > toplevel;
       return(target);
}
             NA_ICONS 2
#define
void
      NA(w,closure.call_data)
Widget
             w;
```

```
closure, call_data;
 caddr_t
 {
        Widget
 shell=ShellWidget("na_shell",(Widget)closure,SW_below,NULL.NULL),
                   form=FormatWidget("na_form".shell), widgets[NA_ICONS];
        Formitem
                     items(] = {
              {"na_confirm"."confirm".0,0,FW_icon,NULL},
              {"na_label", "This function is not available", 0,1,FW_label, NULL},
       };
       XtCallbackRec
                            callbacks[] = {
              {Destroy,(caddr_t)shell}, {NULL,NULL}.
       };
       FillForm(form, NA_ICONS, items, widgets, callbacks);
       XtPopup(shell.XtGrabExclusive);
}
      SetSensitive(w,closure,call_data)
void
Widget
              w;
             closure, call_data;
caddr t
{
      XtSetSensitive((Widget)closure.True);
}
```

source/Process.c

```
Call sub-processes
 */
#include
               "../include/xwave.h"
#include
               <signal.h>
#include
               <sys/wait.h>
#include
               <sys/time.h>
#include
               <sys/resource.h>
/=
              Function Name:
                                    Fork
              Description: Executes a file in a process and waits for termination
                                    argy - standard argy argument description
              Arguments:
              Returns:
                                    dead process id
 */
int
       Fork(argv)
char
       *argv[];
{
              pid;
      int
      union wait
                   statusp;
      struct rusage rusage;
      if (0 = \text{pid} = \text{fork()}) {
             execvp(argv[0],argv);
             exit():
```

```
} else if (pid>0) wait4(pid,&statusp,0,&rusage);
       return(pid);
}
       Function Name:
/=
                            zropen
       Description: Open a file (or .Z file) for reading
                     file_name - name of the file to be read
       Arguments:
                            pid - pointer to process id
       Returns:
                     file pointer
 +/
FILE *zropen(file_name.pid)
       *file_name:
char
       *pid;
int
{
       char z_name(STRLEN);
      String zcat[]={"zcat",z_name,NULL};
       FILE *fp;
      if (NULL = = (fp = fopen(file_name, "r"))) {
              static int
                            up[2];
             sprintf(z_name, "%s.Z", file name);
             pipe(up);
             if (0! = (*pid = fork())) {
                    Dprintf("Parent process started\n");
                    close(up[1]);
                    fp = fdopen(up(0), "r");
             } else {
                    Dprintf("Running zcat on %s\n".zcat[1]);
```

- 275 --

```
close(up[0]);
                     dup2( up[1], 1 );
                     close( up[1] ');
                     execvp(zcat[0],zcat);
              }
       return(fp);
}
/*
       Function Name:
                            zseek
       Description: Fast-forward thru file (fseek will not work on pipes)
       Arguments:
                     fp - file pointer
                            bytes - bytes to skip
void zseek(fp,bytes)
FILE *fp;
int
       bytes;
{
              scratch[1000];
       char
              i;
       int
       while(bytes > 0) {
                     amount = bytes > 1000?1000: bytes;
              int
              fread(scratch.sizeof(char),amount,fp);
              bytes-=amount:
```

```
void zclose(fp.pid)

FILE *fp;
int pid;

{
    union wait statusp;
    struct rusage rusage;

    fclose(fp);
    if (pid!=0) wait4(pid,&statusp,0,&rusage);
}
```

- 277 -

source/PullRightMenu.c

#if (!defined(lint) && !defined(SABER))
static char Xrcsid[] = "\$XConsortium: PullRightMenu.c,v 1.32 89/12/11 15:01:50 kit

#endif

Exp \$":

/*

- * Copyright 1989 Massachusetts Institute of Technology
- * Permission to use, copy, modify, distribute, and sell this software and its
- * documentation for any purpose is hereby granted without fee, provided that
- * the above copyright notice appear in all copies and that both that
- * copyright notice and this permission notice appear in supporting
- * documentation, and that the name of M.I.T. not be used in advertising or
- * publicity pertaining to distribution of the software without specific,
- * written prior permission. M.I.T. makes no representations about the
- * suitability of this software for any purpose. It is provided "as is"
- * without express or implied warranty.
- * M.I.T. DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL
- * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL M.I.T.
- * BE LIABLE FOR ANY SPECIAL. INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
- * WHATSOEVER RESULTING FROM LOSS OF USE. DATA OR PROFITS. WHETHER IN AN ACTION
- * OF CONTRACT. NEGLIGENCE OR OTHER TORTIOUS ACTION. ARISING OUT OF OR IN

- 278 -

```
* CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
*/
* PullRightMenu.c - Source code file for PullRightMenu widget.
#include < stdio.h>
#include <X11/IntrinsicP.h>
#include < X11/StringDefs.h>
#include < X11/Xaw/XawInit.h >
#include < X11/Xaw/SimpleMenP.h>
#include "PullRightMenuP.h"
#include < X11/Xaw/SmeBSB.h>
#include "SmeBSBpr.h"
#include < X11/Xaw/Cardinals.h>
#include < X11/Xmu/Initer.h>
#include < X11/Xmu/CharSet.h >
                      (strcmp((a), (b)) = = 0)
#define streq(a, b)
#define offset(field) XtOffset(PullRightMenuWidget, simple_menu.field)
static XtResource resources[] = {
 * Label Resources.
 */
```

```
{XtNlabel, XtCLabel, XtRString, sizeof(String),
   offset(label_string), XtRString, NULL},
 {XtNlabelClass, XtCLabelClass, XtRPointer, sizeof(WidgetClass),
   offset(label class), XtRImmediate, (caddr_t) NULL},
* Layout Resources.
*/
 {XtNrowHeight, XtCRowHeight, XtRDimension, sizeof(Dimension),
   offset(row height), XtRImmediate, (caddr_t) 0},
 {XtNtopMargin, XtCVerticalMargins, XtRDimension, sizeof(Dimension),
   offset(top_margin), XtRImmediate, (caddr_t) 0},
 {XtNbottomMargin, XtCVerticalMargins, XtRDimension, sizeof(Dimension),
   offset(bottom margin), XtRImmediate, (caddr_t) 0},
/*
* Misc. Resources
*/
 { XtNallowShellResize, XtCAllowShellResize, XtRBoolean, sizeof(Boolean),
    XtOffset(SimpleMenuWidget, shell.allow_shell_resize),
    XtRImmediate, (XtPointer) TRUE },
 {XtNcursor, XtCCursor, XtRCursor, sizeof(Cursor),
    offset(cursor), XtRImmediate, (caddr_t) None},
 {XtNmenuOnScreen, XtCMenuOnScreen, XtRBoolean, sizeof(Boolean),
    offset(menu_on_screen), XtRImmediate, (caddr_t) TRUE},
 {XtNpopupOnEntry, XtCPopupOnEntry, XtRWidget, sizeof(Widget),
    offset(popup_entry), XtRWidget, NULL},
 {XtNbackingStore, XtCBackingStore, XtRBackingStore, sizeof (int),
    offset(backing_store),
     XtRImmediate. (caddr_t) (Always + WhenMapped + NotUseful)}.
```

```
{XtNbutton, XtCWidget, XtRWidget, sizeof(Widget),
        offset(button), XtRWidget, (XtPointer)NULL},
};
#undef offset
static char defaultTranslations[] =
   " < EnterWindow > :
                           highlight()
                                            \ln
    < LeaveWindow > :
                           pull()
                                             \ln
    < BtnMotion > :
                         highlight()
                                          \ln
    <BtnUp>:
                        execute()";
/*
 * Semi Public function definitions.
+/
static void Redisplay(), Realize(), Resize(), ChangeManaged();
static void Initialize(), ClassInitialize(), ClassPartInitialize();
static Boolean SetValues(), SetValuesHook();
static XtGeometryResult GeometryManager();
* Action Routine Definitions
 +/
static void Highlight(), Unhighlight(), Pull(), Execute(), Notify(), PositionMenuAction();
* Private Function Definitions.
static void MakeSetValuesRequest(), CreateLabel(), Layout();
static void AddPositionAction(), PositionMenu(), ChangeCursorOnGrab();
```

```
static Dimension GetMenuWidth(), GetMenuHeight();
static Widget FindMenu();
static SmeObject GetEventEntry();
static XtActionsRec actionsList[] =
                                  Pull},
 {"pull".
                           Execute},
 {"execute",
 {"notify",
                           Notify},
 {"highlight",
                    Highlight},
 {"unhighlight",
                    Unhighlight).
};
CompositeClassExtensionRec pr_extension_rec = {
      /* next_extension */ NULL,
      /* record_type */
                                  NULLQUARK.
      /* version */
                                  XtCompositeExtensionVersion.
      /* record_size */
                                  sizeof(CompositeClassExtensionRec),
      /* accepts_objects */ TRUE,
};
#define superclass (&overrideShellClassRec)
PullRightMenuClassRec pullRightMenuClassRec = {
   /* superclass
                      */ (WidgetClass) superclass.
   /* class_name
                       */
                            "PullRightMenu",
                     */ sizeof(PullRightMenuRec),
   /* size
   /* class initialize */
                           Classinitialize,
   /* class part initialize*/ ClassPartInitialize,
   /* Class init'ed
                           FALSE.
   /* initialize
                         Initialize.
                    */
```

```
/* initialize_hook */ NULL.
                       Realize.
/* realize
                       actionsList,
/* actions
                         XtNumber(actionsList).
/* num_actions
                   */
                        resources.
/* resources
/* resource count
                    */ XtNumber(resources),
                   */ NULLQUARK,
/* xrm_class
/* compress_motion */
                           TRUE.
/* compress_exposure */
                           TRUE,
/* compress_enterleave*/
                               TRUE,
 /* visible_interest */
                       FALSE,
                       NULL.
 /* destroy
                       Resize,
 /* resize
                  */
                   */ Redisplay,
 /* expose
                        SetValues,
 /* set values
                     */ SetValuesHook,
 /* set values_hook
 /* set_values_almost */ XtInheritSetValuesAlmost,
                    */ NULL,
 /* get_values_hook
                    */ NULL,
 /* accept focus
                        XtVersion,
 /* intrinsics version */
                        NULL.
 /* callback offsets */
                          */
                              defaultTranslations,
 /* tm table
                          */ NULL.
 /* query_geometry
 /* display_accelerator*/
                         NULL.
                        NULL
 /* extension
},{
                           GeometryManager,
 /* geometry_manager
 /* change_managed
                           ChangeManaged.
                      XtInheritInsertChild.
 /* insert_child
                       XtInheritDeleteChild.
                   */
 /* delete_child
                   */
                        NULL
 /* extension
},{ ·
```

```
*/ NULL
  /* Shell extension
 }.{
  /* Override extension */
                             NULL
 }.{
  /* Simple Menu extension*/ NULL
 }
};
WidgetClass pullRightMenuWidgetClass = (WidgetClass)&pullRightMenuClassRec:
 * Semi-Public Functions.
      Function Name: ClassInitialize
      Description: Class Initialize routine, called only once.
      Arguments: none.
      Returns: none.
 */
static void
ClassInitialize()
 XawInitializeWidgetSet();
 XtAddConverter( XtRString, XtRBackingStore, XmuCvtStringToBackingStore,
              NULL, 0);
 XmuAddInitializer( AddPositionAction. NULL);
}
       Function Name: ClassInitialize
```

```
Description: Class Part Initialize routine, called for every
                 subclass. Makes sure that the subclasses pick up
                 the extension record.
      Arguments: wc - the widget class of the subclass.
      Returns: none.
*/
static void
ClassPartInitialize(wc)
WidgetClass wc;
{
   SimpleMenuWidgetClass smwc = (SimpleMenuWidgetClass) wc;
/*
 * Make sure that our subclass gets the extension rec too.
 */
   pr extension_rec.next_extension = smwc->composite_class.extension;
   smwc->composite_class.extension = (caddr_t) &pr_extension_rec;
}
       Function Name: Initialize
/+
      Description: Initializes the simple menu widget
       Arguments: request - the widget requested by the argument list.
                       - the new widget with both resource and non
               new
                       resource values.
       Returns: none.
 */
/* ARGSUSED */
static void
Initialize(request. new)
```

```
Widget request, new;
 SimpleMenuWidget smw = (SimpleMenuWidget) new;
 XmuCallInitializers(XtWidgetToApplicationContext(new));
 if (smw->simple_menu.label_class == NULL)
    smw->simple_menu.label_class = smeBSBObjectClass;
 smw->simple menu.label = NULL;
 smw-> simple_menu.entry_set = NULL;
 smw-> simple_menu.recursive_set_values = FALSE;
 if (smw-> simple_menu.label_string != NULL)
    CreateLabel(new);
 smw->simple_menu.menu_width = TRUE;
 if (smw->core.width == 0) {
    smw->simple_menu.menu_width = FALSE;
   smw->core.width = GetMenuWidth(new, NULL);
 }
 smw->simple_menu.menu_height = TRUE;
 if (smw->core.height == 0) {
   smw->simple_menu.menu_height = FALSE;
    smw->core.height = GetMenuHeight(new);
 }
* Add a popup_callback routine for changing the cursor.
```

```
=/
 XtAddCallback(new, XtNpopupCallback, ChangeCursorOnGrab, NULL);
      Function Name: Redisplay
/*
      Description: Redisplays the contents of the widget.
      Arguments: w - the simple menu widget.
              event - the X event that caused this redisplay.
               region - the region the needs to be repainted.
      Returns: none.
 */
/* ARGSUSED */
static void
Redisplay(w, event, region)
Widget w:
XEvent * event;
Region region;
   SimpleMenuWidget smw = (SimpleMenuWidget) w;
   SmeObject * entry:
   SmeObjectClass class:
   if (region = = NULL)
      XClearWindow(XtDisplay(w), XtWindow(w));
   * Check and Paint each of the entries - including the label.
   */
   ForAllChildren(smw. entry) {
```

```
if (!XtIsManaged ( (Widget) *entry)) continue:
      if (region != NULL)
         switch(XRectInRegion(region, (int) (*entry)-> rectangle.x.
                            (int) (*entry)-> rectangle.y,
                            (unsigned int) (*entry)-> rectangle. width,
                            (unsigned int) (*entry)-> rectangle.height)) {
         case RectangleIn:
         case RectanglePart:
             break:
          default:
              continue;
      class = (SmeObjectClass) (*entry)-> object.widget_class;
      if (class->rect_class.expose != NULL)
          (class->rect_class.expose)( (Widget) *entry, NULL, NULL);
   }
}
      Function Name: Realize
/+
      Description: Realizes the widget.
      Arguments: w - the simple menu widget.
               mask - value mask for the window to create.
               attrs - attributes for the window to create.
       Returns: none
 */
static void
Realize(w, mask, attrs)
Widget w:
XtValueMask * mask:
```

```
XSetWindowAttributes * attrs;
{
  SimpleMenuWidget smw = (SimpleMenuWidget) w;
   attrs->cursor = smw->simple_menu.cursor;
   *mask | = CWCursor;
  if ((smw->simple_menu.backing_store == Always) ||
      (smw-> simple_menu.backing_store == NotUseful) ||
      (smw->simple_menu.backing_store == WhenMapped)) {
      *mask |= CWBackingStore:
      attrs->backing_store = smw->simple_memu.backing_store;
   }
   else
      *mask &= ~CWBackingStore;
   (*superclass->core_class.realize) (w, mask, attrs);
}
      Function Name: Resize
/*
      Description: Handle the menu being resized bigger.
      Arguments: w - the simple menu widget.
      Returns: none.
 */
static void
Resize(w)
Widget w:
{
   SimpleMenuWidget smw = (SimpleMenuWidget) w;
   SmeObject * entry;
   if (!XtlsRealized(w)) return:
```

```
ForAllChildren(smw. entry)
                                 /* reset width of all entries. */
      if (XtIsManaged( (Widget) *entry))
         (*entry)-> rectangle.width = smw->core.width;
  Redisplay(w, (XEvent *) NULL, (Region) NULL);
      Function Name: SetValues
/*
      Description: Relayout the menu when one of the resources is changed.
      Arguments: current - current state of the widget.
              request - what was requested.
              new - what the widget will become.
      Returns: none
 */
/* ARGSUSED */
static Boolean
SetValues(current, request, new)
Widget current, request, new;
   SimpleMenuWidget smw_old = (SimpleMenuWidget) current:
   SimpleMenuWidget smw_new = (SimpleMenuWidget) new;
   Boolean ret val = FALSE, layout = FALSE;
   if (!XtlsRealized(current)) return(FALSE);
   if (!smw new-> simple_menu.recursive_set_values) {
       if (smw new->core.width != smw_old->core.width) {
          smw new->simple_menu.menu_width = (smw_new->core.width!= 0);
         layout = TRUE;
       if (smw_new->core.height != smw_old->core.height) {
```

```
smw_new-> simple_menu.menu_height = (smw_new-> core.height != 0);
        layout = TRUE:
  }
  if (smw_old-> simple_menu.cursor != smw_new-> simple_menu.cursor)
     XDefineCursor(XtDisplay(new).
                XtWindow(new), smw_new-> simple_menu.cursor);
  if (smw_old-> simple_menu.label_string != smw_new-> simple_menu.label_string)
      if (smw_new->simple_menu.label_string == NULL)
                                                             /* Destroy. */
         XtDestroyWidget(smw_old-> simple_menu.label);
      else if (smw_old-> simple_menu.label_string = = NULL)
                                                           /* Create. */
         CreateLabel(new);
                                               /* Change. */
      eise {
         Arg args[1];
         XtSetArg(args[0], XtNlabel, smw_new-> simple_menu.label_string);
         XtSetValues(smw_new-> simple_menu.label, args, ONE);
      }
  if (smw_old-> simple_menu.label_class != smw_new-> simple_menu.label_class)
      XtAppWarning(XtWidgetToApplicationContext(new),
                "No Dynamic class change of the SimpleMenu Label.");
  if ((smw_old-> simple_menu.top_margin != smw_new-> simple_menu.top_margin)
11
      (smw_old->simple_menu.bottom_margin!=
      smw new->simple_memu.bottom_margin) /* filler..... */) {
      layout = TRUE;
      ret val = TRUE:
```

```
if (layout)
       Layout(new, NULL, NULL);
   return(ret_vai);
      Function Name: SetValuesHook
/*
      Description: To handle a special case, this is passed the
                 actual arguments.
      Arguments: w - the menu widget.
               arglist - the argument list passed to XtSetValues.
               num args - the number of args.
       Returns: none
 */
/*
 * If the user actually passed a width and height to the widget
 * then this MUST be used, rather than our newly calculated width and
 * height.
 */
static Boolean
SetValuesHook(w, arglist, num_args)
Widget w:
ArgList arglist:
Cardinal *num args;
   register Cardinal i:
    Dimension width, height;
    width = w > core.width:
    height = w->core.height;
```

```
for (i = 0; i < *mum_args; i++)
      if ( streq(arglist[i].name, XtNwidth) )
         width = (Dimension) arglist[i].value;
      if ( streq(arglist[i].name, XtNheight) )
         height = (Dimension) argiist[i].value;
  }
  if ((width != w->core.width) || (height != w->core.height))
      MakeSetValuesRequest(w, width, height);
  return(FALSE);
}
* Geometry Management routines.
      Function Name: GeometryManager
/*
      Description: This is the SimpleMenu Widget's Geometry Manager.
      Arguments: w - the Menu Entry making the request.
              request - requested new geometry.
              reply - the allowed geometry.
       Returns: XtGeometry{Yes, No, Almost}.
 */
static XtGeometryResult
GeometryManager(w, request, reply)
Widget w:
XtWidgetGeometry * request. * reply;
```

/*

```
SimpleMenuWidget smw = (SimpleMenuWidget) XtParent(w);
  SmeObject entry = (SmeObject) w;
  XtGeometryMask mode = request-> request mode;
  XtGeometryResult answer;
  Dimension old height, old width;
  if (!(mode & CWWidth) &&!(mode & CWHeight))
     remm(XtGeometryNo);
  reply-> width = request-> width;
  reply->height = request->height;
  old width = entry-> rectangle.width;
  old height = entry-> rectangle.height;
  Layout(w, &(reply-> width), &(reply-> height));
* Since we are an override shell and have no parent there is no one to
* ask to see if this geom change is okay, so I am just going to assume
* we can do whatever we want. If you subclass be very careful with this
* assumption, it could bite you.
* Chris D. Peterson - Sept. 1989.
*/
  if ( (reply-> width == request-> width) &&
      (reply->height == request->height) ) {
      if ( mode & XtCWQueryOnly ) { /* Actually perform the layout. */
        entry->rectangle.width = old_width;
        entry-> rectangle.height = old_height;
```

+/

```
else {
         Layout(( Widget) smw, NULL, NULL);
      answer = XtGeometryDone;
  }
  else {
      entry-> rectangle.width = old_width;
      entry->rectangle.height = old_height;
      if ( ((reply-> width = = request-> width) && !(mode & CWHeight)) | |
          ((reply->height = = request-> height) && !(mode & CWWidth)) ||
          ((reply-> width == request-> width) &&
           (reply->height == request->height)))
         answer = XtGeometryNo;
      else {
         answer = XtGeometryAlmost;
         reply-> request_mode = 0;
         if (reply-> width != request-> width)
             reply-> request_mode | = CWWidth;
         if (reply->height != request->height)
             reply-> request mode | = CWHeight;
      }
   return(answer);
      Function Name: ChangeManaged
/*
      Description: called whenever a new child is managed.
      Arguments: w - the simple menu widget.
      Returns: none.
```

```
static void
ChangeManaged(w)
Widget w:
   Layout(w, NULL, NULL);
 * Global Action Routines.
 * These actions routines will be added to the application's
 * global action list.
       Function Name: PositionMenuAction
       Description: Positions the simple menu widget.
       Arguments: w - a widget (no the simple menu widget.)
                event - the event that caused this action.
                params. num_params - parameters passed to the routine.
                                we expect the name of the menu here.
        Returns: none
  */
 /* ARGSUSED */
 static void
 PositionMenuAction(w, event, params, num_params)
 Widget w:
 XEvent * event;
  String * params:
  Cardinal * num_params:
```

```
Widget menu;
XPoint loc;
if (*num params!= 1) {
 char error buf[BUFSIZ];
 sprintf(error_buf, "%s %s",
        "Xaw - SimpleMenuWidget: position menu action expects only one",
        "parameter which is the name of the menu.");
 XtAppWarning(XtWidgetToApplicationContext(w), error_buf);
 return:
if ( (menu = FindMenu(w, params[0])) == NULL) {
 char error_buf[BUFSIZ];
 sprintf(error_buf, "%s '%s'",
        "Xaw - SimpleMenuWidget: could not find menu named: ", params[0]);
 XtAppWarning(XtWidgetToApplicationContext(w), error_buf);
 return:
switch (event-> type) {
case ButtonPress:
case ButtonRelease:
  loc.x = event->xbutton.x_root;
  loc.y = event->xbutton.y_root;
  PositionMenu(menu, &loc);
 break:
case EnterNotify:
case LeaveNotify:
  loc.x = event-> xcrossing.x_root:
  loc.y = event-> xcrossing.y root:
```

```
PositionMemu(memu, &loc);
  break:
case MotionNotify:
  loc.x = event-> xmotion.x_root;
  loc.y = event-> xmotion.y_root;
  PositionMenu(menu, &loc);
  break:
default:
  PositionMenu(menu, NULL);
  break:
* Widget Action Routines.
      Function Name: Unhighlight
/+
      Description: Unhighlights current entry.
      Arguments: w - the simple menu widget.
              event - the event that caused this action.
              params, num_params - ** NOT USED **
      Returns: none
/* ARGSUSED */
static void
Unhighlight(w, event, params, num_params)
Widget w:
XEvent * event;
```

```
String * params;
Cardinal * mum params:
   SimpleMenuWidget smw = (SimpleMenuWidget) w;
   SmeObject entry = smw-> simple_menu.entry_set;
   SmeObjectClass class;
   if (entry = = NULL) return;
   smw->simple_menu.entry_set = NULL;
   class = (SmeObjectClass) entry-> object.widget_class;
   (class-> sme_class.unhighlight) ( (Widget) entry);
}
      Function Name: Highlight
      Description: Highlights current entry.
       Arguments: w - the simple menu widget.
               event - the event that caused this action.
               params, num_params - ** NOT USED **
       Returns: none
 */
/* ARGSUSED */
static void
Highlight(w, event, params, num_params)
Widget w;
XEvent * event;
String * params;
Cardinal * num_params;
   SimpleMenuWidget smw = (SimpleMenuWidget) w;
   SmeObject entry;
```

}

/*

```
SmeObjectClass class:
   if (!XtIsSensitive(w)) return;
   entry = GetEventEntry(w, event);
   if (entry = = smw-> simple menu.entry set) return;
   Unhighlight(w, event, params, num params);
   if (entry == NULL) return;
   if (!XtIsSensitive((Widget) entry)) {
      smw-> simple_menu.entry_set = NULL;
      return:
  smw-> simple_memu.entry_set = entry;
  class = (SmeObjectClass) entry-> object.widget class;
   (class-> sme_class.highlight) ( (Widget) entry);
      Function Name: Notify
      Description: Notify user of current entry.
      Arguments: w - the simple menu widget.
              event - the event that caused this action.
              params, num params - ** NOT USED **
      Returns: none
*/
/* ARGSUSED */
```

```
static void
Notify(w, event, params, num_params)
Widget w:
XEvent * event:
String * params:
Cardinal * num_params:
   SimpleMenuWidget smw = (SimpleMenuWidget) w;
   SmeObject entry = smw-> simple_menu.entry_set;
   SmeObjectClass class:
   class = (SmeObjectClass) entry->object.widget_class;
   (class-> sme_class.notify)( (Widget) entry );
}
      Function Name: Pull
      Description: Determines action on basis of leave direction.
      Arguments: w - the pull right menu widget.
              event - the LeaveWindow event that caused this action.
              params, num_params - ** NOT USED **
      Returns: none
  */
 static void Pull(w, event, params, num_params)
             w;
 Widget
             *event:
 XEvent
 String *params;
             *num_params;
 Cardinal
```

```
{
       PullRightMenuWidget
                                   prw=(PullRightMenuWidget)w;
       SmeObject |
                     entry=prw->simple_menu.entry set:
       SmeObjectClass |
                            class:
       if ((entry = = NULL)| | !XtlsSensitive((Widget)entry))return;
       if (event-> type! = LeaveNotify && event-> type! = EnterNotify) {
              XtAppError(XtWidgetToApplicationContext(w),
                "pull() action should only be used with XCrossing events.");
             return:
       if (None! = event-> xcrossing.subwindow) return:
       if (event->xcrossing.y<0 | event->xcrossing.y>prw->core.height) {
              Unhighlight(w,event,params,mm_params);
             return:
       };
       if (event-> xcrossing, x < 0) {
             if (XtIsSubclass(XtParem(w),pullRightMemuWidgetClass)) XtPopdown(w);
             remin:
      }:
   class = (SmeObjectClass)entry-> object.widget_class;
       if (event->xcrossing.x>prw->core.width &&
XtIsSubclass(entry, smeBSBprObjectClass)) (class-> sme_class.notify)((Widget)entry);
      else Unhighlight(w,event,params,num params);
}
/*
      Function Name: Execute
      Description: Determines notify action on basis of SmeObject.
      Arguments: w - the pull right menu widget.
              event - the notify-type event that caused this action.
               params, num params - ** NOT USED **
      Returns: none
```

```
*/
static void Execute(w, event, params, num params)
Widget
              W;
XEvent
              *event:
String *params;
Cardinal
              *mm params;
{
                                  prw = (PullRightMemuWidget)w;
       PullRightMemuWidget
       SmeObject
                    entry = prw- > simple_menu.entry_set;
       SmeObjectClass
                           class:
       Widget
                    shell:
       Dprintf("Execute\n");
       for(shell = w; XtIsSubclass(shell, pullRightMemuWidgetClass); shell = XtParent(shell))
{
              XawSimpleMemrClearActiveEntry(shell);
              XtPopdown(shell);
       };
       if
((entry = = GetEventEntry(w,event))&&(entry! = NULL)&&XtIsSensitive((Widget)entry)) {
              class = (SmeObjectClass)entry-> object.widget class;
              if (XtlsSubclass(entry,smeBSBObjectClass))
(class-> sme_class.notify)((Widget)entry);
       };
```

```
* Public Functions.
      Function Name: XawPullRightMenuAddGlobalActions
/*
      Description: adds the global actions to the simple menu widget.
      Arguments: app_con - the appcontext.
       Returns: none.
*/
void
XawPullRightMenuAddGlobalActions (app\_con)\\
XtAppContext app_con;
{
   XtInitializeWidgetClass(pullRightMenuWidgetClass);
   XmuCallInitializers( app_con );
}
 * Private Functions.
       Function Name: CreateLabel
       Description: Creates a the menu label.
       Arguments: w - the smw widget.
       Returns: none.
 * Creates the label object and makes sure it is the first child in
  * in the list.
```

```
*/
static void
CreateLabel(w)
Widget w;
   SimpleMenuWidget smw = (SimpleMenuWidget) w;
   register Widget * child, * next_child;
   register int i;
   Arg args[2];
   if ((smw->simple_menu.label_string == NULL) ||
        (smw->simple menu.label!= NULL)) {
       char error buf[BUFSIZ];
       sprintf(error_buf, "Xaw Simple Menu Widget: %s or %s, %s",
              "label string is NULL", "label already exists",
              "no label is being created.");
       XtAppWarning(XtWidgetToApplicationContext(w),\ error\_buf);\\
       return;
   }
   XtSetArg(args[0], XtNlabel, smw-> simple_menu.label_string);
   XtSetArg(args[1], XtNjustify, XtJustifyCenter);
    smw->simple_menu.label = (SmeObject)
                        XtCreateManagedWidget("menuLabel",
                                     smw->simple_menu.label_class, w,
                                     args, TWO);
    next_child = NULL;
    for (child = smw->composite.children + smw->composite.num_children,
        i = smw-> composite.num\_children ; i > 0 ; i--, child--) {
```

```
if (next_child != NULL)
         *next_child = *child;
      next_child = child;
   *child = (Widget) smw-> simple_menu.label;
}
       Function Name: Layout
/*
      Description: lays the menu entries out all nice and neat.
       Arguments: w - See below (+++)
               width_ret, height_ret - The returned width and
                                  height values.
       Returns: none.
 * if width == NULL || height == NULL then it assumes the you do not care
 * about the return values, and just want a relayout.
 * if this is not the case then it will set width_ret and height_ret
 * to be width and height that the child would get if it were layed out
 * at this time.
 * +++ "w" can be the simple menu widget or any of its object children.
 */
 static void
 Layout(w, width_ret, height_ret)
 Widget w;
 Dimension *width_ret, *height_ret;
    SmeObject current_entry, *entry;
    SimpleMenuWidget smw;
    Dimension width, height;
```

```
Boolean do_layout = ((height_ret == NULL) || (width_ret == NULL));
Boolean allow_change_size;
height = 0;
if (XtlsSubclass(w, puliRightMenuWidgetClass)) {
    smw = (SimpleMenuWidget) w;
    current entry = NULL;
}
else {
    smw = (SimpleMenuWidget) XtParent(w);
    current_entry = (SmeObject) w;
}
allow_change_size = (!XtIsRealized((Widget)smw) | |
                  (smw-> shell.allow_shell_resize));
if (smw->simple menu.menu_height)
    height = smw->core.height;
 cisc
    if (do layout) {
       height = smw->simple_menu.top_margin;
       ForAllChildren(smw, entry) {
           if (!XtIsManaged( (Widget) *entry)) continue;
           if ( (smw->simple_menu.row_height != 0) &&
              (*entry! = smw-> simple menu.label))
              (*entry)-> rectangle.height = smw-> simple_menu.row_height;
           (*entry)->rectangle.y = height;
           (*entry)->rectangle.x = 0;
           height += (*entry)-> rectangle.height;
        }
```

```
height += smw->simple_menu.bottom_margin;
   }
   else {
      if ((smw->simple_menu.row_height!= 0) &&
          (current_entry != smw-> simple_menu.label) )
          height = smw-> simple_menu.row_height;
   }
if (smw->simple_menu.menu_width)
   width = smw->core.width;
else if (allow_change_size)
   width = GetMenuWidth((Widget) smw, (Widget) current_entry);
else
   width = smw->core.width;
if (do_layout) {
   ForAllChildren(smw, entry)
      if (XtIsManaged((Widget) *entry))
          (*entry)-> rectangle.width = width;
   if (allow_change_size)
      MakeSetValuesRequest((Widget) smw, width, height);
}
else {
    *width_ret = width;
   if (height !=0)
       *height ret = height;
}
```

/* Function Name: AddPositionAction

}

* Description: Adds the XawPositionSimpleMenu action to the global

```
action list for this appcon.
      Arguments: app_con - the application context for this app.
              data - NOT USED.
      Returns: none.
*/
/* ARGSUSED */
static void
AddPositionAction(app_con, data)
XtAppContext app_con;
caddr_t data;
{
   static XtActionsRec pos_action[] = {
      { "XawPositionSimpleMenu", PositionMenuAction },
   };
   XtAppAddActions(app_con, pos_action, XtNumber(pos_action));
}
       Function Name: FindMenu
/*
       Description: Find the menu give a name and reference widget.
       Arguments: widget - reference widget.
               name - the menu widget's name.
       Returns: the menu widget or NULL.
 */
 static Widget
 FindMenu(widget, name)
 Widget widget;
 String name;
    register Widget w, menu;
```

```
for ( w = widget; w!= NULL; w = XtParent(w))
      if ( (menu = XtNameToWidget(w, name)) != NULL )
         return(menu);
   return(NULL);
}
      Function Name: PositionMenu
      Description: Places the menu
      Arguments: w - the simple menu widget.
              location - a pointer the the position or NULL.
       Returns: none.
 */
static void
PositionMenu(w, location)
Widget w;
XPoint * location;
{
   SimpleMenuWidget smw = (SimpleMenuWidget) w;
   SmeObject entry;
   XPoint t_point;
   static void MoveMenu();
   if (location == NULL) {
       Window junk1, junk2;
       int root_x, root_y, junkX, junkY;
       unsigned int junkM;
       location = &t_point;
       if (XQueryPointer(XtDisplay(w), XtWindow(w), &junk1, &junk2,
                      &root_x, &root_y, &junkX, &junkY, &junkM) == FALSE) {
```

}

/*

```
char error_buf[BUFSIZ];
      sprintf(error_buf, "%s %s", "Xaw - SimpleMenuWidget:",
             "Could not find location of mouse pointer");
      XtAppWarning(XtWidgetToApplicationContext(w),\ error\_buf);
      return;
   location > x = (short) root_x;
   location > y = (short) root_y;
}
 * The width will not be correct unless it is realized.
 */
XtRealizeWidget(w);
location > x -= (Position) w->core.width/2;
if (smw->simple_menu.popup_entry == NULL)
    entry = smw-> simple_menu.label;
else
    entry = smw->simple_menu.popup_entry;
if (entry != NULL)
    location->y -= entry->rectangle.y + entry->rectangle.height/2;
MoveMenu(w, (Position) location->x, (Position) location->y);
    Function Name: MoveMenu
    Description: Actually moves the menu, may force it to
              to be fully visable if menu_on_screen is TRUE.
```

```
Arguments: w - the simple menu widget.
              x, y - the current location of the widget.
      Returns: none
*/
static void
MoveMenu(w, x, y)
Widget w;
Position x, y;
   Arg arglist[2];
   Cardinal num_args = 0;
   SimpleMenuWidget smw = (SimpleMenuWidget) w;
   if (smw->simple_menu.menu_on_screen) {
       int width = w->core.width + 2 * w->core.border_width;
       int height = w->core.height + 2 * w->core.border_width;
       if (x < 0)
          x = 0;
       else {
          int scr_width = WidthOfScreen(XtScreen(w));
          if (x + width > scr_width)
              x = scr width - width;
       }
       if (y < 0)
           y = 0;
       else {
           int scr_height = HeightOfScreen(XtScreen(w));
           if (y + height > scr_height)
               y = scr height - height;
```

```
}
  XtSetArg(arglist[num_args], XtNx, x); num_args++;
  XtSetArg(arglist[num_args], XtNy, y); num_args++;
  XtSetValues(w, arglist, num_args);
}
/*
      Function Name: ChangeCursorOnGrab
      Description: Changes the cursor on the active grab to the one
                 specified in out resource list.
       Arguments: w - the widget.
               junk, garbage - ** NOT USED **.
       Returns: None.
 +/
/* ARGSUSED */
static void
ChangeCursorOnGrab(w, junk, garbage)
Widget w;
caddr t junk, garbage;
{
   SimpleMenuWidget smw = (SimpleMenuWidget) w;
   /*
    * The event mask here is what is currently in the MIT implementation.
    * There really needs to be a way to get the value of the mask out
    * of the toolkit (CDP 5/26/89).
    */
    X Change A \`ctive Pointer Grab (Xt Display (w), Button Press Mask | Button Release Mask,
                         smw->simple_menu.cursor, CurrentTime);
```

```
}
      Function Name: MakeSetValuesRequest
/*
      Description: Makes a (possibly recursive) call to SetValues,
                I take great pains to not go into an infinite loop.
      Arguments: w - the simple menu widget.
              width, height - the size of the ask for.
      Returns: none
 */
static void
MakeSetValuesRequest(w, width, height)
Widget w;
Dimension width, height;
{
   SimpleMenuWidget smw = (SimpleMenuWidget) w;
   Arg arglist[2];
   Cardinal num_args = (Cardinal) 0;
   if (!smw->simple_menu.recursive_set_values) {
       if ( (smw->core.width != width) || (smw->core.height != height) ) {
          smw->simple_menu.recursive_set_values = TRUE;
          XtSetArg(arglist[num_args], XtNwidth, width); num_args++;
          XtSetArg(arglist[mm_args], XtNheight, height); num_args++;
          XtSetValues(w, arglist, num_args);
       }
       else if (XtIsRealized( (Widget) smw))
          Redisplay((Widget) smw, (XEvent *) NULL, (Region) NULL);
    }
    smw->simple_menu.recursive_set_values = FALSE;
 }
```

```
Function Name: GetMenuWidth
/*
      Description: Sets the length of the widest entry in pixels.
      Arguments: w - the simple menu widget.
      Returns: width of menu.
 */
static Dimension
GetMenuWidth(w, w ent)
Widget w, w_ent;
{
   SmeObject cur_entry = (SmeObject) w_ent;
   SimpleMenuWidget smw = (SimpleMenuWidget) w;
   Dimension width, widest = (Dimension) 0;
   SmeObject * entry;
   if ( smw-> simple_menu.menu_width )
       return(smw->core.width);
   ForAllChildren(smw, entry) {
       XtWidgetGeometry preferred;
       if (!XtIsManaged( (Widget) *entry)) continue;
        if (*entry != cur_entry) {
           XtQueryGeometry(*entry, NULL, &preferred);
           if (preferred.request_mode & CWWidth)
               width = preferred width;
           else
               width = (*entry)-> rectangle.width;
        else
```

```
width = (*entry)-> rectangle.width;
      if ( width > widest )
         widest = width;
  }
  return(widest);
}
      Function Name: GetMenuHeight
/+
      Description: Sets the length of the widest entry in pixels.
      Arguments: w - the simple menu widget.
      Returns: width of menu.
 */
static Dimension
GetMenuHeight(w)
Widget w;
   SimpleMenuWidget smw = (SimpleMenuWidget) w;
   SmeObject * entry;
   Dimension height;
   if (smw->simple_menu.menu_height)
       return(smw->core.height);
   height = smw->simple_menu.top_margin + smw->simple_menu.bottom_margin;
    if (smw->simple_menu.row_height == 0)
       ForAllChildren(smw, entry)
          if (XtIsManaged ((Widget) *entry))
              height += (*entry)->rectangle.height;
```

WO 94/23385

```
else
      height += smw->simple_menu.row_height * smw->composite.num_children;
  return(height);
      Function Name: GetEventEntry
      Description: Gets an entry given an event that has X and Y coords.
      Arguments: w - the simple menu widget.
              event - the event.
      Returns: the entry that this point is in.
*/
static SmeObject
GetEventEntry(w, event)
Widget w;
XEvent * event;
  Position x_loc, y_loc;
   SimpleMenuWidget smw = (SimpleMenuWidget) w;
   SmeObject * entry;
   switch (event-> type) {
  case MotionNotify:
      x_{loc} = event-> xmotion.x;
      y loc = event-> xmotion.y;
      break;
  case EnterNotify:
  case LeaveNotify:
      x loc = event-> xcrossing.x;
      y_loc = event->xcrossing.y;
      break:
```

```
case ButtonPress:
case ButtonRelease:
   x loc = event-> xbutton.x;
   y_loc = event->xbutton.y;
    break;
default:
    XtAppError(XtWidgetToApplicationContext(w),\\
             "Unknown event type in GetEventEntry().");
    break;
}
if ( (x_loc < 0) \mid | (x_loc > = smw-> core.width) | | (y_loc < 0) | |
    (y_loc > = smw-> core.height))
    return(NULL);
 ForAllChildren(smw, entry) {
    if (!XtIsManaged ((Widget) *entry)) continue;
    if ( ((*entry)-> rectangle.y < y_loc) &&
        ((*entry)-> rectangle.y + (*entry)-> rectangle.height > y_loc))
        if ( *entry == smw->simple_menu.label )
                              /* cannot select the label. */
            return(NULL);
        else
            return(*entry);
 }
 remrn(NULL);
```

source/Select.c

```
/*
* Selection from list widget
 */
              "../include/xwave.h"
#include
       Select(w,closure,call_data)
biov
Widget
              w;
              closure, call_data;
caddr t
{
                     sel = (Selection) closure;
       Selection
                     button=FindWidget(sel->button,w),
       Widget
                     shell = ShellWidget(sel-> name,button,SW_below,NULL,NULL),
                     form=FormatWidget("sel_form",shell), list_widget, widgets[3];
       String *list=(sel->list_proc)();
                     items[] = {
        Formitem
               {"sel_cancel", "close", 0, 0, FW_icon, NULL},
               {"sel_label",(String)sel->action_name,1,0,FW_label,NULL},
               {"sel_view",NULL,0,2,FW_view,NULL},
        };
                            list_calls[] = {
        XtCallbackRec
               {Destroy,(caddr_t)shell},
               {sel->action_proc,sel->action_closure},
               {NULL, NULL},
       . }, callbacks[] = {
```

```
- 319 -
```

```
{Destroy,(caddr_t)shell},
{NULL,NULL},
};
Arg args[1];

FillForm(form,THREE,items,widgets,callbacks);
XtSetArg(args[0],XtNlist,list);

list_widget = XtCreateManagedWidget("sel_list",listWidgetClass,widgets[2],args,ONE);
XtAddCallbacks(list_widget,XtNcallback,list_calls);
XtPopup(shell,XtGrabExclusive);
}
```

source/SmeBSBpr.c

```
#if (!defined(lint) && !defined(SABER) )
static char Xrcsid[] = "$XConsortium: SmeBSB.c,v 1.9 89/12/13 15:42:48 kit Exp $";
#endif
```

/+

* Copyright 1989 Massachusetts Institute of Technology

*

- * Permission to use, copy, modify, distribute, and sell this software and its
- * documentation for any purpose is hereby granted without fee, provided that
- * the above copyright notice appear in all copies and that both that
- * copyright notice and this permission notice appear in supporting
- * documentation, and that the name of M.I.T. not be used in advertising or
- * publicity pertaining to distribution of the software without specific,
- * written prior permission. M.I.T. makes no representations about the
- * suitability of this software for any purpose. It is provided "as is"
- * without express or implied warranty.

*

- * M.I.T. DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL
- * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL M.I.T.
- * BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
- * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION
- * OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN
- * CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

```
*/
* SmeBSBpr.c - Source code file for BSB pull-right Menu Entry object.
*/
#include < stdio.h>
#include < X11/IntrinsicP.h >
#include <X11/StringDefs.h>
#include <X11/Xmu/Drawing.h>
#include < X11/Xaw/XawInit.h>
#include < X11/Xaw/SimpleMenu.h>
#include "SmeBSBprP.h"
#include <X11/Xaw/Cardinals.h>
#define ONE_HUNDRED 100
#define offset(field) XtOffset(SmeBSBprObject, sme_bsb.field)
static XtResource resources[] = {
  {XtNlabel, XtCLabel, XtRString, sizeof(String),
    offset(label), XtRString, NULL},
  {XtNvertSpace, XtCVertSpace, XtRInt, sizeof(int),
    offset(vert_space), XtRImmediate, (caddr_t) 25},
  {XtNleftBitmap, XtCLeftBitmap, XtRPixmap, sizeof(Pixmap),
    offset(left_bitmap), XtRImmediate, (caddr_t)None},
  {XtNjustify, XtCJustify, XtRJustify, sizeof(XtJustify),
    offset(justify), XtRImmediate, (caddr_t) XtJustifyLeft},
  {XtNrightBitmap, XtCRightBitmap, XtRPixmap, sizeof(Pixmap),
```

```
offset(right_bitmap), XtRImmediate, (caddr_t)None},
 {XtNleftMargin, XtCHorizontalMargins, XtRDimension, sizeof(Dimension),
   offset(left_margin), XtRImmediate, (caddr_t) 4},
 {XtNrightMargin, XtCHorizontalMargins, XtRDimension, sizeof(Dimension),
   offset(right_margin), XtRImmediate, (caddr_t) 4},
 {XtNforeground, XtCForeground, XtRPixel, sizeof(Pixel),
   offset(foreground), XtRString, "XtDefaultForeground"},
 {XtNfont, XtCFont, XtRFontStruct, sizeof(XFontStruct *),
   offset(font), XtRString, "XtDefaultFont"},
 {XtNmenuName, XtCMenuName, XtRString, sizeof(String),
       offset(menu_name), XtRString, (caddr_t)"menu"},
};
#undef offset
* Semi Public function definitions.
*/
static void Redisplay(), Destroy(), Initialize(), FlipColors(), PopupMenu();
static void ClassInitialize();
static Boolean SetValues();
static XtGeometryResult QueryGeometry();
 * Private Function Definitions.
 */
static void GetDefaultSize(), DrawBitmaps(), GetBitmapInfo();
static void CreateGCs(), DestroyGCs();
#define superclass (&smeClassRec)
SmeBSBprClassRec smeBSBprClassRec = {
```

```
(WidgetClass) superclass,
/* superclass
                        "SmeBSBpr",
/ class_name
                      sizeof(SmeBSBprRec).
/* size
/* class_initializer */
                       ClassInitialize,
/* class_part_initialize*/ NULL,
                       FALSE,
/* Class init'ed
                      Initialize,
/* initialize
                      NULL,
/* initialize_hook */
/* realize
                       NULL,
                       NULL,
/* actions
                         ZERO,
/* num actions
                        resources,
/* resources
                    */ XtNumber(resources),
/* resource count
                        NULLQUARK,
/* xrm_class
                           FALSE,
/* compress motion
                           FALSE,
/* compress_exposure */
/* compress_enterleave*/
                               FALSE,
/* visible_interest */ FALSE,
                   */
                        Destroy,
 /* destroy
                       NULL,
 /* resize
                   */
                        Redisplay,
 /* expose
                        SetValues,
/* set_values
                     */ NULL,
 /* set values hook
 /* set_values_almost */ XtInheritSetValuesAlmost,
                      */ NULL,
 /* get values_hook
                          NULL,
 /* accept focus
                         XtVersion,
 /* intrinsics version */
                         NULL,
 /* callback offsets */
                                NULL,
                           */
 /* tm table
                                QueryGeometry,
                           */
 /* query_geometry
                          NULL,
  /* display accelerator*/
```

```
*/ NULL
  /* extension
}.{
  /* Menu Entry Fields */
                        FlipColors,
  /* highlight */
  /* unhighlight */
                        FlipColors,
  /* notify */
                         PopupMenu,
  /* extension
                    */ NULL
 }, {
  /* BSB pull-right Menu entry Fields */
  /* extension
                     */ NULL
 }
};
WidgetClass smeBSBprObjectClass = (WidgetClass) &smeBSBprClassRec;
 * Semi-Public Functions.
      Function Name: ClassInitialize
      Description: Initializes the SmeBSBprObject.
      Arguments: none.
      Returns: none.
 */
static void
ClassInitialize()
```

```
XawInitializeWidgetSet();
  XtAddConverter( XtRString, XtRJustify, XmuCvtStringToJustify, NULL, 0 );
}
      Function Name: Initialize
/*
      Description: Initializes the simple menu widget
      Arguments: request - the widget requested by the argument list.
                      - the new widget with both resource and non
               new
                      resource values.
      Returns: none.
 */
/* ARGSUSED */
static void
Initialize(request, new)
Widget request, new;
{
   SmeBSBprObject entry = (SmeBSBprObject) new;
   if (entry->sme_bsb.label == NULL)
       entry-> sme_bsb.label = XtName(new);
   else
       entry->sme_bsb.label = XtNewString(entry->sme_bsb.label);
       /* Xaw bug - bitmap initialization now performed */
    if (entry-> sme_bsb.left_bitmap! = None) GetBitmapInfo(entry, TRUE);
    if (entry->sme_bsb.right_bitmap!= None) GetBitmapInfo(entry, FALSE);
    CreateGCs(new);
    GetDefaultSize(new, &(entry->rectangle.width), &(entry->rectangle.height));
```

```
Function Name: Destroy
/*
      Description: Called at destroy time, cleans up.
      Arguments: w - the simple menu widget.
      Returns: none.
 */
static void
Destroy(w)
Widget w;
   SmeBSBprObject entry = (SmeBSBprObject) w;
    DestroyGCs(w);
   if (entry->sme_bsb.label != XtName(w))
       XtFree(entry-> sme_bsb.label);
 }
       Function Name: Redisplay
 /*
       Description: Redisplays the contents of the widget.
       Arguments: w - the simple menu widget.
                event - the X event that caused this redisplay.
                region - the region the needs to be repainted.
        Returns: none.
  */
 /* ARGSUSED */
 static void
 Redisplay(w, event, region)
 Widget w;
 XEvent * event;
 Region region;
```

```
GC gc;
SmeBSBprObject entry = (SmeBSBprObject) w;
int font ascent, font_descent, y_loc;
entry->sme_bsb.set_values_area_cleared = FALSE;
font ascent = entry-> sme_bsb.font-> max_bounds.ascent;
font_descent = entry-> sme_bsb.font-> max_bounds.descent;
y loc = entry->rectangle.y;
if (XtIsSensitive(w) && XtIsSensitive( XtParent(w) ) ) {
   if ( w == XawSimpleMenuGetActiveEntry(XtParent(w)) ) {
       XFillRectangle(XtDisplayOfObject(w), XtWindowOfObject(w),
                   entry-> sme_bsb.norm_gc, 0, y_loc,
                   (unsigned int) entry->rectangle.width,
                    (unsigned int) entry->rectangle.height);
      gc = entry-> sme_bsb.rev_gc;
    }
    else
       gc = entry->sme_bsb.norm_gc;
}
else
    gc = entry-> sme_bsb.norm_gray_gc;
if (entry-> sme_bsb.label != NULL) {
    int x loc = entry-> sme_bsb.left_margin;
    int len = strlen(entry-> sme_bsb.label);
    char * label = entry-> sme_bsb.label;
    switch(entry->sme bsb.justify) {
       int width, t_width;
```

/*

```
case XtJustifyCenter:
     t_width = XTextWidth(entry->sme_bsb.font, label, len);
     width = entry-> rectangle.width - (entry-> sme_bsb.left_margin +
                                   entry-> sme_bsb.right_margin);
     x_{loc} += (width - t_width)/2;
     break:
  case XtJustifyRight:
      t_width = XTextWidth(entry->sme_bsb.font, label, len);
      x_loc = entry-> rectangle.width - (entry-> sme_bsb.right_margin +
                                   t_width);
      break;
   case XtJustifyLeft:
   default:
      break:
   }
   y_loc += (entry->rectangle.height -
           (font_ascent + font_descent)) / 2 + font_ascent;
   XDrawString(XtDisplayOfObject(w), XtWindowOfObject(w), gc,
             x_loc, y_loc, label, len);
}
DrawBitmaps(w, gc);
   Function Name: SetValues
   Description: Relayout the menu when one of the resources is changed.
   Arguments: current - current state of the widget.
            request - what was requested.
            new - what the widget will become.
```

```
Returns: none
*/
/* ARGSUSED */
static Boolean
SetValues(current, request, new)
Widget current, request, new;
   SmeBSBprObject entry = (SmeBSBprObject) new;
   SmeBSBprObject old_entry = (SmeBSBprObject) current;
   Boolean ret_val = FALSE;
   if (old_entry->sme_bsb.label != entry->sme_bsb.label) {
      if (old_entry->sme_bsb.label != XtName( new ) )
          XtFree( (char *) old_entry-> sme_bsb.label );
       if (entry->sme_bsb.label != XtName(new))
          entry->sme_bsb.label = XtNewString(entry->sme_bsb.label);
       ret_val = True;
   }
    if (entry-> rectangle.sensitive != old_entry-> rectangle.sensitive )
       ret val = TRUE;
    if (entry-> sme_bsb.left_bitmap != old_entry-> sme_bsb.left_bitmap) {
       GetBitmapInfo(new, TRUE);
        ret_val = TRUE;
    }
    if (entry->sme_bsb.right_bitmap != old_entry->sme_bsb.right_bitmap) {
       GetBitmapInfo(new, FALSE);
```

```
ret val = TRUE;
  }
  if ( (old_entry-> sme_bsb.font != entry-> sme_bsb.font) | |
      (old_entry->sme_bsb.foreground != entry->sme_bsb.foreground) ) {
      DestroyGCs(current);
      CreateGCs(new);
      ret val = TRUE;
  }
  if (ret_val) {
      GetDefaultSize(new,
                   &(entry-> rectangle.width), &(entry-> rectangle.height));
      entry->sme_bsb.set_values_area_cleared = TRUE;
   return(ret_val);
      Function Name: QueryGeometry.
/*
      Description: Returns the preferred geometry for this widget.
      Arguments: w - the menu entry object.
               itended, return_val - the intended and return geometry info.
       Returns: A Geometry Result.
* See the Intrinsics manual for details on what this function is for.
 * I just return the height and width of the label plus the margins.
 */
static XtGeometryResult
QueryGeometry(w, intended, return_val)
Widget w;
```

```
XtWidgetGeometry *intended, *return_val;
{
   SmeBSBprObject entry = (SmeBSBprObject) w;
   Dimension width, height;
   XtGeometryResult ret_val = XtGeometryYes;
   XtGeometryMask mode = intended-> request_mode;
   GetDefaultSize(w, &width, &height);
   if ( ((mode & CWWidth) && (intended-> width != width)) ||
       !(mode & CWWidth) ) {
       return_val->request_mode | = CWWidth;
       return_val->width = width;
       ret_val = XtGeometryAlmost;
   }
   if ( ((mode & CWHeight) && (intended->height != height)) | |
        !(mode & CWHeight) ) {
       return_val->request_mode |= CWHeight;
       return_val->height = height;
       ret val = XtGeometryAlmost;
    }
    if (ret_val = = XtGeometryAlmost) {
       mode = return_val-> request_mode;
       if ( ((mode & CWWidth) && (width == entry-> rectangle.width)) &&
           ((mode & CWHeight) && (height == entry->rectangle.height)))
          return(XtGeometryNo);
    }
    return(ret_val);
```

```
}
      Function Name: FlipColors
/*
      Description: Invert the colors of the current entry.
      Arguments: w - the bsb menu entry widget.
      Returns: none.
*/
static void
FlipColors(w)
Widget w;
{
   SmeBSBprObject entry = (SmeBSBprObject) w;
   if (entry-> sme_bsb.set_values_area_cleared) return;
   XFillRectangle(XtDisplayOfObject(w),\ XtWindowOfObject(w),
                entry-> sme_bsb.invert_gc, 0, (int) entry-> rectangle.y,
                (unsigned int) entry->rectangle.width,
                (unsigned int) entry->rectangle.height);
}
 * Private Functions.
       Function Name: GetDefaultSize
/*
       Description: Calculates the Default (preferred) size of
                 this menu entry.
       Arguments: w - the menu entry widget.
```

```
width, height - default sizes (RETURNED).
      Returns: none.
*/
static void
GetDefaultSize(w, width, height)
Widget w;
Dimension * width, * height;
   SmeBSBprObject entry = (SmeBSBprObject) w;
   if (entry-> sme_bsb.label == NULL)
       *width = 0;
   clse
       *width = XTextWidth(entry-> sme_bsb.font, entry-> sme_bsb.label,
                       strlen(entry-> sme_bsb.label));
   *width += entry->sme_bsb.left_margin + entry->sme_bsb.right_margin;
   *height = (entry->sme_bsb.font->max_bounds.ascent +
            entry->sme_bsb.font->max_bounds.descent);
    *height = (*height * ( ONE_HUNDRED +
                      entry->sme_bsb.vert_space )) / ONE_HUNDRED;
 }
       Function Name: DrawBitmaps
 /*
       Description: Draws left and right bitmaps.
       Arguments: w - the simple menu widget.
                gc - graphics context to use for drawing.
       Returns: none
```

```
static void
DrawBitmaps(w, gc)
Widget w;
GC gc;
   int x_loc, y_loc;
   SmeBSBprObject entry = (SmeBSBprObject) w;
   if ( (entry-> sme_bsb.left_bitmap == None) &&
        (entry->sme_bsb.right_bitmap == None) ) return;
/*
 * Draw Left Bitmap.
 */
   y_{loc} = entry-> rectangle.y + (entry-> rectangle.height -
                             entry-> sme_bsb.left_bitmap_height) / 2;
  if (entry-> sme_bsb.left_bitmap != None) {
   x_loc = (entry->sme_bsb.left_margin -
           entry->sme_bsb.left_bitmap_width) / 2;
    X CopyPlane (XtDisplayOfObject (w), entry-> sme\_bsb.left\_bitmap,
             XtWindowOfObject(w), gc, 0, 0,
             entry-> sme_bsb.left_bitmap_width,
             entry-> sme_bsb.left_bitmap_height, x_loc, y_loc, 1);
  }
  * Draw Right Bitmap.
  */
    y_loc = entry->rectangle.y + (entry->rectangle.height - /* Xaw bug - y_loc
```

```
calculated from right_bitmap data */
                             entry-> sme_bsb.right_bitmap_height) / 2;
 if (entry->sme_bsb.right_bitmap! = None) {
   x_loc = entry-> rectangle.width - (entry-> sme_bsb.right_margin + /* Xaw bug - +
rather than - sign */
                                entry->sme_bsb.right_bitmap_width) / 2;
   XCopyPlane(XtDisplayOfObject(w), entry-> sme_bsb.right_bitmap,
            XtWindowOfObject(w), gc, 0, 0,
            entry-> sme_bsb.right_bitmap_width,
            entry-> sme_bsb.right_bitmap_height, x_loc, y_loc, 1);
 }
}
       Function Name: GetBitmapInfo
/*
       Description: Gets the bitmap information from either of the bitmaps.
       Arguments: w - the bsb menu entry widget.
               is left - TRUE if we are testing left bitmap,
                        FALSE if we are testing the right bitmap.
       Returns: none
 */
 static void
 GetBitmapInfo(w, is_left)
 Widget w;
 Boolean is left;
    SmeBSBprObject entry = (SmeBSBprObject) w;
    unsigned int depth, bw;
    Window root:
    int x, y;
    unsigned int width, height;
```

```
char buf[BUFSIZ];
if (is_left) {
   if (entry-> sme_bsb.left_bitmap != None) {
       if (!XGetGeometry(XtDisplayOfObject(w),
                      entry-> sme_bsb.left_bitmap, &root,
                      &x, &y, &width, &height, &bw, &depth)) {
           sprintf(buf, "SmeBSB Object: %s %s \"%s\".", "Could not",
                 "get Left Bitmap geometry information for menu entry ",
                 XtName(w));
           XtAppError(XtWidgetToApplicationContext(w),\ buf);
       }
      if (depth != 1)
           sprintf(buf, "SmeBSB Object: %s \"%s\"%s.",
                  "Left Bitmap of entry ",
                  XtName(w), " is not one bit deep.");
           XtAppError(XtWidgetToApplicationContext(w), buf);
       }
       entry-> sme_bsb.left_bitmap_width = (Dimension) width;
       entry-> sme_bsb.left_bitmap_height = (Dimension) height;
    }
 }
 else if (entry-> sme_bsb.right_bitmap != None) {
     if (!XGetGeometry(XtDisplayOfObject(w),
                    entry-> sme_bsb.right_bitmap, &root,
                    &x, &y, &width, &height, &bw, &depth)) {
        sprintf(buf, "SmeBSB Object: %s %s \"%s\".", "Could not",
               "get Right Bitmap geometry information for memu entry ",
               XtName(w));
        XtAppError(XtWidgetToApplicationContext(w), buf);
     }
     if (depth != 1) {
```

```
sprintf(buf, "SmeBSB Object: %s \"%s\"%s.",
                *Right Bitmap of entry ", XtName(w),
                " is not one bit deep.");
         XtAppError(XtWidgetToApplicationContext(w), buf);
      }
      entry->sme_bsb.right_bitmap_width = (Dimension) width;
      entry->sme_bsb.right_bitmap_height = (Dimension) height;
  }
}
/*
      Function Name: CreateGCs
      Description: Creates all gc's for the simple menu widget.
      Arguments: w - the simple menu widget.
      Returns: none.
 */
static void
CreateGCs(w)
Widget w;
{
   SmeBSBprObject entry = (SmeBSBprObject) w;
   XGCValues values;
   XtGCMask mask;
   values.foreground = XtParent(w)->core.background_pixel;
   values.background = entry-> sme_bsb.foreground;
   values.font = entry-> sme_bsb.font-> fid;
   values.graphics_exposures = FALSE;
              = GCForeground | GCBackground | GCFont | GCGraphicsExposures;
   mask
   entry->sme_bsb.rev_gc = XtGetGC(w, mask, &values);
    values.foreground = entry-> sme_bsb.foreground;
```

```
values.background = XtParent(w)->core.background_pixel;
  entry->sme_bsb.norm_gc = XtGetGC(w, mask, &values);
  values.fill_style = FillTiled;
  values.tile = XmuCreateStippledPixmap(XtScreenOfObject(w),
                                   entry-> sme bsb.foreground,
                                   XtParent(w)->core.background_pixel,
                                   XtParent(w)->core.depth);
  values.graphics_exposures = FALSE;
  mask |= GCTile | GCFillStyle;
  entry->sme_bsb.norm_gray_gc = XtGetGC(w, mask, &values);
  values.foreground ^= values.background;
  values.background = 0;
  values.function = GXxor;
  mask = GCForeground | GCBackground | GCGraphicsExposures | GCFunction;
  entry->sme_bsb.invert_gc = XtGetGC(w, mask, &values);
/*
      Function Name: DestroyGCs
      Description: Removes all gc's for the simple menu widget.
      Arguments: w - the simple menu widget.
      Returns: none.
*/
static void
DestroyGCs(w)
Widget w;
{
   SmeBSBprObject entry = (SmeBSBprObject) w;
   XtReleaseGC(w, entry-> sme_bsb.norm_gc);
```

```
XtReleaseGC(w, entry->sme_bsb.norm_gray_gc);
  XtReleaseGC(w, entry-> sme_bsb.rev_gc);
  XtReleaseGC(w, entry-> sme_bsb.inven_gc);
}
#ifdef apollo
/*
* The apollo compiler that we have optomizes out my code for
* FlipColors() since it is static. and no one executes it in this
 * file. I am setting the function pointer into the class structure so
 * that it can be called by my parent who will tell me to when to
 * highlight and unhighlight.
 */
void XawSmeBSBApolloHack ()
   FlipColors();
#endif /* apollo */
/* Hacked copy of PopupMenu from MenuButton widget to replace XUnheritNotify */
static void
PopupMenu(w, event, params, num_params)
Widget w;
XEvent * event;
String * params;
Cardinal * num_params;
  SmeBSBprObject mbw = (SmeBSBprObject) w;
  Widget menu, temp;
```

```
Arg arglist[2];
Cardinal num args;
int menu_x, menu_y, menu_width, menu_height, button_width, button_height;
Position button_x, button_y;
temp = XtParent(w); /* Shell not menu entry is parent of menu */
while(temp != NULL) {
 menu = XtNameToWidget(temp, mbw->sme_bsb.menu_name);
 if (menu == NULL)
   temp = XtParent(temp);
 else
   break:
if (menu == NULL) {
 char error buf[BUFSIZ];
 sprintf(error_buf, "MenuButton: %s %s.",
        "Could not find menu widget named", mbw->sme_bsb.menu_name);
 XtAppWarning(XtWidgetToApplicationContext(w), error_buf);
 return:
if (!XtIsRealized(menu))
  XtRealizeWidget(menu);
menu_width = menu->core.width + 2 * menu->core.border_width;
button width = w->core.width + 2 * w->core.border_width;
button_height = w->core.height + 2 * w->core.border_width;
menu height = menu->core.height + 2 * menu->core.border_width;
XtTranslateCoords(w, 0, 0, &button_x, &button_y);
menu_x = button_x + button_width;
```

```
menu y = button_y;
if (menu_x < 0)
 menu_x = 0;
else {
 int scr_width = WidthOfScreen(XtScreen(menu));
 if (menu_x + menu_width > scr_width)
   menu_x = scr_width - menu_width;
}
if (menu_y < 0)
  menu_y = 0;
else {
  int scr height = HeightOfScreen(XtScreen(menu));
 if (menu_y + menu_height > scr_height)
   menu_y = scr_height - menu_height;
}
num_args = 0;
XtSetArg(arglist[num_args], XtNx, menu_x); num_args++;
XtSetArg(arglist[num_args], XtNy, menu_y); num_args++;
XtSetValues(menu, arglist, num_args);
XtPopupSpringLoaded(menu);
```

```
source/Storage.c
```

```
/*
      Routines to allow video frames to be stored in memory
      or on disk: NewFrame, GetFrame, SaveFrame, FreeFrame, SaveHeader,
CopyHeader.
*/
             "../include/xwave.h"
#include
extern FILE *zropen();
extern void
             zseek();
extern void
             zclose();
       NewFrame(vid,number)
void
Video vid;
int
       number;
{
       if (vid-> data[0][number] = = NULL) {
                    channel, channels=vid->type==MONO?1:3;
             int
              for(channel=0;channel<channels;channel++)
                    vid->data[channel][number]=(short
*)MALLOC(sizeof(short)*Size(vid,channel,0)*Size(vid,channel,1));
}
       GetFrame(vid,number)
 void
 Video vid:
```

```
number;
int
{
      if (vid-> data[0][number] = = NULL) {
                   file_name[STRLEN], *whole_frame;
            FILE *fp, *fopen();
                   pid, r, c, channel,
             int
                          start = vid - > x_offset + vid - > cols*vid - > y_offset,
end = (vid-> rows-vid-> y_offset-vid-> size[1])*vid-> cols-vid-> x_offset,
                          inter=vid->cols-vid->size[0];
             NewFrame(vid, number);
iles[0] = = '\0'?vid-> name:vid-> files,number + vid-> start);
             Dprintf("Reading file %s\n",file_name);
             fp=zropen(file_name,&pid);
             if (vid->precision = =0) whole_frame = (char
 *)MALLOC(vid->rows*vid->cols);
              zseek(fp, vid-> offset);
              for(channel = 0; channel < (vid-> type = = MONO?1:3); channel + +) {
                           shift[2] = {vid-> type = = YUV &&
 channel! = 0?vid-> UVsample[0]:0,vid-> type = = YUV &&
 channel! = 0?vid-> UVsample[1]:0};
                     Dprintf("Reading channel %d\n",channel);
                     if (vid->precision==0) {
 if (0 = = fread(whole_frame, size of (char), (vid > cols > > shift[0])*(vid > rows > > shift[1]),
 fp)) {
                                  Dprintf("Error whilst reading %s\n",file_name);
```

```
- 344 -
```

```
Eprintf("Error whilst reading %s\n",file_name);
                                                                                           }
                                                                                           for(r=0;r < vid-> size[1] >> shift[1];r++)
                                                                                                                  for(c=0;c < vid-> size[0] > > shift[0];c++) {
pel = cti(whole\_frame[(vid->x\_offset>> shift[0]) + c + ((vid->y\_offset>> shift[1]) + r)*(
vid > cols > shift[0]);
\label{eq:continuous_vid} vid-> data[channel][number][c+r*(vid-> size[0] >> shift[0])] = vid-> negative?-1-pel:pel;
                                                                       } else {
                                                                                              if (start! =0) zseek(fp,start*sizeof(short));
                                                                                              for(r=0;r < vid-> size[1]>> shift[1];r++) {
   if(0 = = fread(\&(vid-> data[channel][number][r+(vid-> size[0] > > shift[0])]), size of(short),
    vid-> size[0] > > shift[0], fp)) {
                                                                                                                                             Dprintf("Error whilst reading
     %s\n",file_name);
                                                                                                                                              Eprintf("Error whilst reading
     %s\n",file_name);
                                                                                                                        if (inter! = 0) zseek(fp,inter*sizeof(short));
                                                                                                                        if (vid-> negative)
                                                                                                                                               for(c=0;c < vid-> size[0] >> shift[0];c++)
      \label{eq:channel} vid-> data[channel][number][c+r*(vid->size[0]>>shift[0])] = -1-vid->data[channel][number][c+r*(vid->size[0]>>shift[0])] = -1-vid->data[channel][number][c+r*(vid->size[0]>>shift[0])] = -1-vid->data[channel][number][c+r*(vid->size[0]>>shift[0])] = -1-vid->data[channel][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][numbe
      mber][c+r*(vid->size[0]>>shift[0])];
```

source/Storage.c

```
/*
      Routines to allow video frames to be stored in memory
      or on disk: NewFrame, GetFrame, SaveFrame, FreeFrame, SaveHeader,
CopyHeader.
*/
             "../include/xwave.h"
#include
extern FILE *zropen();
extern void zseek();
extern void zclose();
      NewFrame(vid,number)
void
Video vid;
int
       number;
{
       if (vid-> data[0][number] = = NULL) {
                    channel, channels=vid->type==MONO?1:3;
              int
              for(channel=0;channel<channels;channel++)
                    vid->data[channel][number]=(short
 *)MALLOC(sizeof(short)*Size(vid,channel,0)*Size(vid,channel,1));
       }
 }
       GetFrame(vid, number)
 void
 Video vid;
```

```
number;
int
{
       if (vid->data[0][number] = = NULL) {
                     file name[STRLEN], *whole_frame;
              FILE *fp, *fopen();
                     pid, r, c, channel,
              int
                            start = vid -> x_offset + vid -> cols*vid -> y_offset,
end = (vid-> rows-vid-> y\_offset-vid-> size[1])*vid-> cols-vid-> x\_offset,
                            inter=vid->cols-vid->size[0];
              NewFrame(vid, number);
sprintf(file\_name, "%s%s/%s/%s/%s%03d\0", global->home, IMAGE\_DIR, vid->path, vid->f
iles[0] = = '\0'?vid-> name:vid-> files, number + vid-> start);
              Dprintf("Reading file %s\n",file_name);
              fp=zropen(file name,&pid);
              if (vid->precision = =0) whole_frame = (char
*)MALLOC(vid->rows*vid->cols);
              zseek(fp, vid->offset);
              for(channel = 0; channel < (vid-> type = = MONO?1:3); channel + +) {
                            shift[2] = {vid-> type = = YUV &&
                     int
channel!=0?vid->UVsample[0]:0,vid->type==YUV &&
channel! = 0?vid-> UVsample[1]:0};
                     Dprintf("Reading channel %d\n",channel);
                     if (vid-> precision = = 0) {
if(0 = fread(whole frame, size of(char), (vid > cols > shift[0])*(vid > rows > shift[1]),
fp)) {
                                   Dprintf("Error whilst reading %s\n",file name);
```

```
- 347 -
```

```
Eprintf("Error whilst reading %s\n",file_name);
                                                                                                                                                           }
                                                                                                                                                           for(r=0;r < vid-> size[1] >> shift[1];r++)
                                                                                                                                                                                                   for(c=0;c < vid-> size[0] >> shift[0];c++) 
                                                                                                                                                                                                                                           short
pel = cti(whole\_frame\{(vid->x\_offset>> shift[0]) + c + ((vid->y\_offset>> shift[1]) + r) + ((vid->y\_offset>> shift[1]) +
 vid->cols>>shift[0])]);
  \label{eq:vid-data} vid-> data[channel][number][c+r*(vid->size[0]>>shift[0])] = vid->negative?-1-pel:pel;
                                                                                                                        } else {
                                                                                                                                                               if (start! = 0) zseek(fp,start*sizeof(short));
                                                                                                                                                                for(r=0;r < vid-> size[1] >> shift[1];r++) 
     if (0 = = fread(\&(vid-> data[channel][number][r^+(vid-> size[0] > > shift[0])]), size of (short),
     vid-> size[0] > > shift[0],fp)) {
                                                                                                                                                                                                                                               Dprintf("Error whilst reading
       %s\n", file name);
                                                                                                                                                                                                                                                Eprintf("Error whilst reading
        %s\n",file_name);
                                                                                                                                                                                                          }
                                                                                                                                                                                                         if (inter! = 0) zseek(fp,inter*sizeof(short));
                                                                                                                                                                                                          if (vid-> negative)
                                                                                                                                                                                                                                                 for(c=0;c < vid-> size[0] > > shift[0];c++)
         \label{eq:channel} vid-> data[channel][number][c+r*(vid->size[0]>>shift[0])] = -1-vid->data[channel][number][c+r*(vid->size[0]>>shift[0])] = -1-vid->data[channel][number][c+r*(vid->size[0]>>shift[0])] = -1-vid->data[channel][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][number][num
          mber][c+r^*(vid-> size[0] > > shift[0])];
```

```
}
       SaveHeader(vid)
void
Video vid:
{
       FILE *fp, *fopen();
              file name[STRLEN];
       char
       String types[]={"MONO","RGB","YUV"};
       Dprintf("SaveHeader %s\n", vid-> name);
sprintf(file_name, "%s%s/%s%s\0",global->home,VID_DIR,vid->name,VID_EXT);
        fp = fopen(file_name, "w");
        fprintf(fp, "Path \" %s\"\n", vid-> path);
        if (vid-> files[0]! = '\0') fprintf(fp, "Files \" %s\"\n", vid-> files);
        if (vid->type==YUV) fprintf(fp, "Type %s %d
 %d\n^*, types[vid-> type], vid-> UVsample[0], vid-> UVsample[1]);
        else fprintf(fp, "Type %s\n", types[vid->type]);
        if (vid->rate!=0) fprintf(fp, "Rate %d\n", vid->rate);
        if (vid->disk) fprintf(fp, "Disk\n");
        if (vid->gamma) fprintf(fp, "Gamma\n");
        fprintf(fp, "Start %03d\n", vid-> start);
        fprintf(fp, "Length %d\n", vid-> size[2]);
        fprintf(fp, "Dimensions %d %d\n", vid->cols, vid->rows);
        switch(vid->trans.type) {
               TRANS_None: fprintf(fp, "Transform None\n"); break;
        case
               TRANS_Wave: fprintf(fp, "Transform Wavelet %d %d
  %s\n",vid->trans.wavelet.space[0],vid->trans.wavelet.space[1],vid->trans.wavelet.dirn
  ?"Yes":"No"); break;
```

```
}
      fprintf(fp, "Header %d\n", vid-> offset);
      fprintf(fp, "Offsets %d %d\n", vid-> x_offset, vid-> y_offset);
      fprintf(fp, "Size %d %d\n", vid->size[0], vid->size[1]);
      fprintf(fp, "Precision %d\n", vid->precision);
      fclose(fp);
}
Video CopyHeader(src)
Video src;
{
       Video dst=(Video)MALLOC(sizeof(VideoRec));
              channel;
       int
       Dprintf("CopyHeader %s\n",src);
       strcpy(dst->path,src->path);
       strepy(dst->name,src->name);
       dst - type = src - type;
       dst-> disk=src-> disk;
       dst-> gamma = src-> gamma;
       dst-> negative = False;
        dst-> rate = src-> rate;
        dst-> start = src-> start;
       dst-> size[0] = src-> size[0];
        dst-> size[1] = src-> size[1];
        dst-> size[2] = src-> size[2];
        dst->UVsample[0]=src->UVsample[0];
        dst->UVsample[1]=src->UVsample[1];
        dst-> offset=0;
        dst->cols=src->size[0];
```

```
- 350 -
```

source/Transform.c

```
/*
      Transform video using wavelet transform
*/
#include
              "xwave.h"
              "Transform.h"
#include
extern short Round();
       DropVideo(w,closure,call_data)
void
Widget
              w;
              closure, cali_data;
caddr t
 {
       Video video=global->videos->next;
              frame, channel;
       int
 for(channel = 0; channel < (global-> videos-> type = = MONO?1:(global-> videos-> type =
 = YUV?3:4));channel++)
              if (global->videos->data[channel]!=NULL) {
                      for (frame = 0; frame < global-> videos-> size[2]; frame + +)
                            if (global->videos->data[channel][frame]!=NULL)
 XtFree(global->videos->data[channel][frame]);
                      XtFree(global->videos->data[channel]);
        XtFree(global-> videos);
        global-> videos = video;
```

```
}
      ChangePrecision(src,dst,frame,old,new)
void
Video src, dst;
      frame, old, new;
int
{
      int
            channel, i;
      if(src! = dst || old! = new) {
                  shift = new-old;
            int
            Dprintf("Changing precision %d to %d for frame %d\n",old,new,frame);
            for (channel=0;channel<(src->type==MONO?1:3);channel++) {
                         size = Size(src,channel,0)*Size(src,channel,1);
                   int
                   for(i=0; i < size; i++)
= = 0?src-> data[channel][frame][i]:src-> data[channel][frame][i] < < shift);
            }
      }
}
      TransformCtrl(w,closure,call_data)
void
Widget
             closure, call_data;
caddr t
{
                   ctrl=(TransCtrl)closure;
       TransCtrl
```

```
Video src = ctrl > src, dst = CopyHeader(src);
     long i, frame, channel;
     Dprintf("TransformCtrl\n");
      strcpy(dst->name,ctrl->name);
      dst->trans.type=TRANS_Wave;
      dst-> trans.wavelet.space[0] = ctrl-> space[0];
      dst-> trans.wavelet.space[1] = ctri-> space[1];
      dst->trans.wavelet.dirn=ctrl->dirn;
      dst->precision=ctrl->precision;
      strcpy(dst->files,dst->name);
      if (dst->disk) SaveHeader(dst);
      if (src->trans.type!=TRANS_Wave) {
             src-> trans.type = TRANS_Wave;
             src-> trans.wavelet.space[0]=0;
             src-> trans.wavelet.space[1]=0;
      }
      if (src-> trans.wavelet.space[0]!=dst-> trans.wavelet.space[0] | |
src-> trans.wavelet.space[1]!=dst-> trans.wavelet.space[1])
             for(frame = 0; frame < dst-> size[2]; frame + +) {
                    int
max_precision=src->precision>dst->precision?src->precision:dst->precision;
                    Dprintf("Processing frame %d\n",frame);
                     NewFrame(dst, frame);
                     GetFrame(src,frame);
                     ChangePrecision(src,dst,frame,src-> precision,max_precision);
                     for (channel = 0; channel < (src-> type = = MONO?1:3); channel + +)
{
                                   oct src=src->trans.wavelet.space[channel==0?0:1],
                            int
```

```
oct_{dst} = dst - > trans.wavelet.space[channel = = 0?0:1],
size[2] = {Size(dst,channel,0),Size(dst,channel,1)};
                             if (oct_src! = oct_dst)
Convolve(dst->data[channel][frame],ctrl->dirn,size,oct_src,oct_dst);
                      ChangePrecision(dst,dst,frame,max_precision,dst->precision);
                      SaveFrame(dst,frame);
                      FreeFrame(dst,frame);
                      FreeFrame(src,frame);
               }
        if (src-> trans.wavelet.space[0] = = 0 && src-> trans.wavelet.space[1] = = 0)
src-> trans.type = TRANS_None;
        if (dst->trans.wavelet.space[0] = = 0 && dst->trans.wavelet.space[1] = = 0) {
               dst-> trans.type = TRANS_None;
               if (dst->disk) SaveHeader(dst);
        }
        dst->next=global->videos;
        global-> videos = dst;
 }
        Transtype(w,closure,call_data)
 void
 Widget
                w;
                ciosure, call_data;
 caddr_t
 {
         Video vid = (Video) closure;
         if (vid-> trans.wavelet.space[0] = = 0 && vid-> trans.wavelet.space[1] = = 0)
```

```
vid->trans.type=TRANS_None;
}
void BatchTransCtrl(w,closure,call_data)
Widget
             closure, call_data;
caddr t
{
                    ctrl=(TransCtrl)closure;
       TransCtrl
       if (ctrl->src = NULL) ctrl->src=FindVideo(ctrl->src_name,global->videos);
       if (ctrl->src->trans.type = = TRANS_Wave)
ctrl->dirn=ctrl->src->trans.wavelet.dirn;
       TransformCtrl(w,closure,call_data);
}
             InitTransCtrl(name)
TransCtri
String name;
{
                    ctrl = (TransCtrl)MALLOC(sizeof(TransCtrlRec));
       TransCtrl
       strcpy(ctrl->src_name,name);
       strcpy(ctrl->name,name);
       ctrl->dim=False;
       Dprintf("Transform\n");
       return(ctrl);
 }
                                  16
              TRANS_ICONS
 #define
```

```
Transform(w,closure,call_data)
void
Widget
             w;
             closure, call data;
caddr t
{
       Video video = (Video)closure;
                     ctrl=InitTransCtrl(video-> name);
       TransCtri
                     spaceInput = (NumInput) MALLOC (2*size of (NumInputRec)), \\
       NumInput
                            precInput=(NumInput)MALLOC(sizeof(NumInputRec));
                     msg = NewMessage(ctrl-> name, NAME_LEN);
       Message
                            destroy call[]={
       XtCallbackRec
              {Free,(caddr_t)ctrl},
              {Free,(caddr_t)spaceInput},
              {Free,(caddr_t)precInput},
              {CloseMessage,(caddr_t)msg},
              {NULL, NULL},
       };
                     parent = FindWidget("frm_transform", XtParent(w)),
       Widget
shell = ShellWidget("transform", parent, SW_below, NULL.destroy_call),
                     form=FormatWidget("trans_form",shell),
 widgets[TRANS_ICONS];
                     items[] = {
       FormItem
               {"trans_cancel", "cancel", 0, 0, FW_icon, NULL},
               {"trans confirm", "confirm", 1,0,FW_icon, NULL},
               {"trans_title", "Transform a video", 2, 0, FW_label, NULL},
               {"trans_vid_lab", "Video Name: ",0,3,FW_label,NULL},
               {"trans video", NULL, 4, 3, FW_text, (String) msg},
               {"trans_dirn_lab", "Direction: ",0,4,FW_label,NULL},
               {"trans_dirn".NULL,4,4,FW_yn,(String)&ctrl->dirn},
```

```
{"trans bits_int", NULL, 0, 6, FW_integer, (String) precInput},
      {"trans bits_down", NULL, 4, 6, FW_down, (String) precInput},
      {"trans_bits_up", NULL, 9, 6, FW_up, (String) precInput},
      {"trans_spc0_int", NULL, 0, 8, FW_integer, (String)&spaceInput[0]},
      {"trans_spc0_down", NULL, 4, 8, FW_down, (String) & spaceInput[0]},
      {"trans_spc0_up", NULL, 12, 8, FW_up, (String) & spaceInput[0]},
      {"trans_spc1_int", NULL, 0, 11, FW_integer, (String) & spaceInput[1]},
      {"trans_spc1_down", NULL, 4, 11, FW_down, (String) & spaceInput[1]},
       {"trans_spc1_up",NULL,15,11,FW_up,(String)&spaceInput[1]},
};
                     callbacks[] = {
XtCallbackRec
       {Destroy,(caddr_t)shell},
       {NULL, NULL},
       {TransformCtrl,(caddr_t)ctrl},
       {Destroy,(caddr_t)shell},
       {NULL, NULL},
       {ChangeYN,(caddr_t)&ctrl->dirn}, {NULL,NULL},
       {NumIncDec,(caddr_t)precInput}, {NULL,NULL},
       \{NumIncDec,(caddr\_t)precInput\},\ \{NULL,NULL\},
       \{NumIncDec, (caddr\_t) \& spaceInput[0]\}, \ \{NULL, NULL\},
       \label{eq:numincDec,(caddr_t)&spaceInput[0]}, $$\{NULL,NULL\}, $$
       \label{eq:numincDec,(caddr_t)&spaceInput[1]}, $$ NULL, NULL$, $$
       \label{eq:numincDec,(caddr_t)&spaceInput[1]}, $$ NULL, NULL$, $$
};
Dprintf("Transform\n");
msg->rows=1; msg->cols=NAME_LEN;
ctrl-> src = video;
if (video-> trans.type = = TRANS_Wave) {
       ctrl-> space[0] = video-> trans. wavelet.space[0];
```

```
ctrl-> space[1] = video-> trans.wavelet.space[1];
            ctrl->dim=video->trans.wavelet.dim;
      } else {
            ctrl-> space[0]=0; ctrl-> space[1]=0;
            ctrl->dim=False;
     }
      ctrl->precision=video->precision;
      spaceInput[0].format=video->type==YUV?"Y-Space: %d":"Space: %d";
      spaceInput[0].max = 100;
      spaceInput[0].min=0;
      spaceInput[0].value = &ctrl-> space[0];
      if (video-> type = = YUV) {
             spaceInput[1].format = "UV-Space: %d";
             spaceInput[1].max = 100;
             spaceInput[1].min=0;
             spaceInput[1].value = &ctrl-> space[1];
      }
      precInput-> format = "Precision: %d";
      precInput - > max = 16;
      precInput-> min=0;
      precInput-> value = &ctrl-> precision;
FillForm(form, TRANS\_ICONS-(video->type==YUV?0:3), items, widgets, callbacks);
      if (video->trans.type==TRANS_Wave) XtSetSensitive(widgets[6],False);
      XtPopup(shell, XtGrabExclusive);
```

source/Update.c

```
/*
       Update Image, Info and InfoText from positional information
*/
              "../include/xwave.h"
#include
               < varargs.h>
#include
              CompositePixel();
extern int
              Dither();
extern int
extern short Round();
              ReMap();
extern int
extern Palette
                      FindPalette();
       *ResizeData(size)
char
       size;
int
                      *data = NULL;
       static char
                      data_size=0;
        static int
        if (size! = data_size) {
               Dprintf("New frame memory\n");
               if (data!=NULL) XtFree(data);
               data=(char *)MALLOC(size);
               data_size=size;
        }
        return(data);
 }
```

```
UpdateImage(frame)
Pixmap
Frame frame;
{
      int
             x, y, i;
                    *dpy=XtDisplay(global->toplevel);
      Display
      void CvtIndex(), UpdatePoint();
                    pal = FindPalette(global-> palettes, frame-> palette);
      Palette
       Video vid=frame->video;
             scrn=XDefaultScreen(dpy), depth=DisplayPlanes(dpy,scrn),
       int
                    size[2] = {Size(vid, frame-> channel, 0), Size(vid, frame-> channel, 1)},
                    img_size[2] = {size[0] < < frame-> zoom, size[1] < < frame-> zoom},
                    bpl=(img_size[0]*depth+7)/8, new_size=img_size[1]*bpl,
                    space=vid->trans.wavelet.space[vid->type==YUV &&
frame->channel!=0 && frame->channel!=3?1:0];
              *data=ResizeData(new size);
       char
       XImage
*image = XCreateImage(dpy,global-> visinfo-> visual,depth,ZPixmap,0,data,img_size[0],i
mg size[1],8,bpl);
       Pixmap
pixmap = XCreatePixmap(dpy,DefaultRootWindow(dpy),img_size[0],img_size[1],depth);
       Dprintf("UpdateImage\n");
       if (global->levels = = 2 && frame->channel = = 3) frame->channel = 0;
       for(y=0;y < size[1];y++) for(x=0;x < size[0];x++) {
                     data x=x, data_y=y, off_x, off_y, oct;
              int
              if (vid->trans.type = = TRANS_Wave)
 CvtIndex(x,y,size[0],size[1],space,&data_x,&data_y,&oct);
              for(off_x=0;off_x<1<<frame->zoom;off_x++)
                     for(off_y=0;off_y<1<<frame->zoom;off_y++) {
```

```
img_x = off_x + (x < frame > zoom),
                              int
img_y = off_y + (y < frame- > zoom),
pix = CompositePixel(frame,data_x,data_y,img_x,img_y);
XPutPixel(image,img_x,img_y,ReMap(pix,global->levels,pal));
       }
XPutImage(dpy,pixmap,DefaultGC(dpy,scrn),image,0,0,0,0,img_size[0],img_size[1]);
        if (frame->point_switch==True) UpdatePoint(dpy,frame,pixmap);
        XtFree(image);
        return(pixmap);
}
        CvtIndex(x,y,max_x,max_y,oct,ret_x,ret_y,ret_oct)
void
        x, y, max_x, max_y, oct, *ret_x, *ret_y, *ret_oct;
 int
 {
                       hgx=x>=(max_x>>1), hgy=y>=(max_y>>1);
        Boolean
        ret_x = hgx?x-(max_x > > 1):x;
        *ret y = hgy?y-(max_y > > 1):y;
        if (!hgx && !hgy && oct>1) {
 \label{eq:cvtIndex} \text{CvtIndex}(\text{*ret}_x,\text{*ret}_y,\text{max}_x >> 1,\text{max}_y >> 1,\text{oct-}1,\text{ret}_x,\text{ret}_y,\text{ret}_\text{oct});
                *ret_x = *ret_x < < 1;
                *ret_y= *ret_y < < 1;
                *ret oct += 1;
         } else {
```

```
ret_x = (ret_x < < 1) + hgx;
                                               *ret y=(\text{*ret } y < < 1) + \text{hgy};
                                               *ret oct = hgx | hgy?0:1;
                       }
}
                        UpdateInfo(frame)
void
 Frame frame;
 {
                                                                        msg=frame->msg;
                         Message
                         Video vid=frame-> video;
                                                 *locn = frame-> point-> location, posn[2] = {locn[0],locn[1]},
                         int
                                                                        channel = 3 = = frame-> channel?0:frame-> channel,
  width = Size(vid, channel, 0);
                          short *data = vid- > data[channel][frame- > frame];
                         msg-> info.ptr[0] = '\0';
                          msg-> info.length=0;
                         if (vid->type==YUV && channel!=0) {
                                                 posn(0) = posn(0) > vid-> UVsample(0);
                                                 posn[1] = posn[1] > vid-> UVsample[1];
                          }
                          if (vid-> trans.type! = TRANS_Wave)
                                                 Mprintf(msg, "Point : x = \%03d y = \%03d t = \%03d
  c = \%4d", locn[0], locn[1], frame-> frame+vid-> start, data[posn[0]+Size(vid, channel, 0)*postart, data[posn
   sn[1]]);
                          else {
                                                                         octs = vid-> trans. wavelet.space[vid-> type = = YUV &&
                                                  int
   channel! = 0?1:0],
                                                                                                 X, Y, oct, sub,
```

```
blkDC[2] = {(posn[0] > octs)\&-2,(posn[1] > octs)\&-2},
                            offDC[2] = \{(posn[0] > octs)\&1, (posn[1] > octs)\&1\};
              Mprintf(msg, "Point : f = \%03d = \%03d
y = \%03d\n^*, frame -> frame + vid -> start, locn[0], locn[1]);
              Mprintf(msg, "Low pass: x = \%03d y = \%03d \ n", blkDC[0], blkDC[1]);
              for(Y=0;Y<2;Y++) {
                     for(X=0;X<2;X++)
Mprintf(msg, "\%4d\%c", data[Access(blkDC[0] + X, blkDC[1] + Y, octs-1, 0, width)], X = = off
DC[0] \&\& Y = = offDC[1]?'*':');
                      Mprintf(msg, "\n");
               }
               for(oct=octs;oct>0;oct-) {
                             blk[2] = \{(posn[0] > > oct) \& -2, (posn[1] > > oct) \& -2\},\
                      int
                                    off[2] = {(posn[0] > oct)&1,(posn[1] > oct)&1};
                      Mprintf(msg, "Oct: %d\n", oct);
                      for(Y=0;Y<2;Y++) {
                             for(sub=1;sub<4;sub++) {
                                     for(X=0;X<2;X++) {
 Mprintf(msg, "%4d%c", data[Access(blk[0] + X, blk[1] + Y, oct-1, sub, width)], X = = off[0]
  && Y = = off[1]?'*':');
                                     }
                                     if (sub < 3) Mprintf(msg,"
                              }
                              if (oct! = 0 \mid | Y = = 0) Mprintf(msg, "\n");
                       }
                }
         }
```

```
Mflush(msg);
}
       Function Name:
                            CrossHair-
/*
                    Draws cross-hair on pixmap
      Description:
                     dpy - Xserver display
       Arguments:
                            pixmap - pixmap to draw on
                            gc - GC to draw with
                            x_off, y_off - offset into pixmap
                            width, height - size of box containing cross-hair
                            x, y - coordinates within box
                            zoom - scaling factor
       Returns:
                     alters pixmap.
 */
       CrossHair(dpy,pixmap,gc,x_off,y_off,width,height,x,y,zoom)
void
Display
              *dpy;
Pixmap
              pixmap;
GC -
       x off, y off, width, height, x, y, zoom;
int
       int
              xtra = Shift(1, zoom);
       x_off=Shift(x_off,zoom);
       y_off=Shift(y_off,zoom);
       width=Shift(width,zoom);
       height = Shift(height,zoom);
       x = Shift(x,zoom);
       y = Shift(y,zoom);
```

```
XFillRectangle(dpy,pixmap,gc,x+x_off+xtra/2,y_off,1,y); /* North hair */
      XFillRectangle(dpy,pixmap,gc,x_off,y+y_off+xtra/2.x,1); /* West hair */
      XFillRectangle(dpy,pixmap,gc,x+x_off+xtra/2,y+y_off+xtra,1,height-y-xtra); /*
South hair */
      XFillRectangle(dpy,pixmap,gc,x+x\_off+xtra,y+y\_off+xtra/2,width-x-1,1);/*
East hair */
}
                           UpdatePoint
       Function Name:
/*
       Description: Draws cross-hair on image at frame->location
                    dpy - X server display
       Arguments:
                           frame - Frame supplying drawing parameters
                           pixmap - X pixmap to draw on
                    alters pixmap.
       Returns:
 */
       UpdatePoint(dpy,frame,pixmap)
void
Display
              *dpy;
Frame frame;
Pixmap
              pixmap;
 {
       unsigned long
                            gcmask;
       XGCValues gcvals;
        GC
              gc:
        Video vid=frame->video;
              posn[2] = \{frame-> point-> location[0], frame-> point-> location[1]\},
 channel = 3 = = frame->channel?0:frame->channel;
        gcvals.function=GXequiv;
        gcmask=GCFunction;
```

```
gcvals.foreground = 127;
      gcmask = gcmask | GCForeground;
      gc = XCreateGC(dpy,pixmap,gcmask,&gcvals);
      if (vid->type==YUV && channel!=0) {
             posn[0] = posn[0] > vid-> UVsample[0];
             posn[1] = posn[1] > vid-> UVsample[1];
      if (vid->trans.type!=TRANS Wave) {
CrossHair(dpy,pixmap,gc,0,0,Size(vid,channel,0),Size(vid,channel,1),posn[0],posn[1],fra
me->zoom);
      } else {
                    octs = vid-> trans. wavelet.space[vid-> type = = YUV &&
             int
channel! = 0?1:0], oct,
                           size[2] = {Size(vid,channel,0),Size(vid,channel,1)};
CrossHair(dpy,pixmap,gc,0,0,size[0],size[1],posn[0],posn[1],frame->zoom-octs);
             for(oct=1;oct < = octs;oct++) {
CrossHair(dpy,pixmap,gc,size[0],0,size[0],size[1],posn[0],posn[1],frame->zoom-oct);
CrossHair(dpy,pixmap,gc,0,size[1],size[0],size[1],posn[0],posn[1],frame->zoom-oct);
CrossHair(dpy,pixmap,gc,size[0],size[1],size[0],size[1],posn[0],posn[1],frame->zoom-oct
);
              }
       XFreeGC(dpy,gc);
}
```

source/Video2.c

```
/*
       Video callback routines for Listing, Loading
*/
               "../include/xwave.h"
#include
               "../include/ImageHeader.h"
#include
               "../include/DTheader.h"
#include
               "Video.h"
#include
               < sys/time.h>
#include
extern void
               EraseFrame();
extern void
               CvtIndex();
void
        SortList(list,no)
String list[];
int
        no:
{
        int
               i, j, k;
        if (no > 1) for(i=1; i < no; i++) for(j=0; j < i; j++) {
                k=0;
                while (list[i][k] = = list[j][k] && list[i][k]! = '\0' && list[j][k]! = '\0') k++;
                if (list[i][k] < list[j][k]) {
                       String spare=list[i];
                       list[i] = list[j];
                       list[j] = spare;
                }
```

```
}
}
String *ReadDirectory(dir_path.extension)
String dir path, extension;
{
              *dirp, *opendir();
       DIR
       struct dirent *dp, *readdir();
       static String *fileList=NULL, file;
              count=0, i;
       int
              path[STRLEN];
       char
       Dprintf("ReadDirectory for %s extension\n", extension);
       if (fileList! = NULL) {
              for(i=0;NULL!=fileList[i];i++) free(fileList[i]);
              free(fileList);
       }
       fileList = (String *)MALLOC(sizeof(String *)*300);
       sprintf(path, "%s%s\0", global-> home, dir_path);
       dirp = opendir(path);
       for (dp=readdir(dirp);dp!=NULL && count < 299;dp=readdir(dirp)) {
                      length=strlen(dp->d_name);
              int
              if (length > = strlen(extension))
               if (!strcmp(dp->d_name+length-strlen(extension),extension)) {
                      Dprintf("Found %s in dir\n",dp->d_name);
                      fileList[count] = (char *)MALLOC(length+1);
                      strncpy(fileList[count],dp->d_name,length-strlen(extension));
                      count + = 1:
               }
```

```
}
       fileList[count] = NULL;
       SortList(fileList,count);
       closedir(dirp);
       return(fileList);
}
int
       Shift(value, shift)
       value, shift;
int
{
       if (shift==0) return value;
       else if (shift < 0) return(value > > -shift);
       else return(value < < shift);
}
       Size(video, channel, dimension)
int
Video video;
       channel, dimension;
int
{
       if (video->type==YUV && dimension!=2 && channel!=0 && channel!=3)
return(video-> size[dimension] > > video-> UV sample[dimension]);
       else remm(video->size[dimension]);
}
        Address2(video,channel,x,y)
int
 Video video;
        channel, x, y;
 int
```

}

```
- 370 -
```

```
{
      if (video->type==YUV && channel!=0 && channel!=3)
remm(x + Size(video, channel, 0) * y);
      else return(x+video->size(0)*y);
}
      Address(video,channel,x,y)
int
Video video;
int
      channel, x, y;
{
      if (video-> type = = YUV && channel! = 0 && channel! = 3)
return((x > video-> UVsample[0]) + Size(video, channel, 0) *(y > video-> UVsample[1])
);
      else return(x+video-> size[0]*y);
}
String *VideoList()
{
      Dprintf("VideoList\n");
      return(ReadDirectory(VID_DIR,VID_EXT));
}
String *KlicsList()
{
      Dprintf("KlicsList\n");
      return(ReadDirectory(KLICS_DIR,KLICS_EXT));
```

```
String *KlicsListSA()
{
      Dprintf("KlicsListSA\n");
      return(ReadDirectory(KLICS_SA_DIR,KLICS_SA_EXT));
}
String *VideoCurrentList()
{
       static String videoList[300];
       Video video=global->videos;
              count = 0;
       int
       Dprintf("VideoCurrentList\n");
       while (video! = NULL) {
              if (count = = 300) Dprintf("VideoCurrentList: static size exceeded\n");
              videoList[count] = video- > name;
              video = video -> next;
              count + = 1;
       }
       videoList[count] = NULL;
       SortList(videoList,count);
       return(videoList);
}
 String *VideoYUVList()
 {
        static String videoList[300];
        Video video=global->videos;
        int
               count=0;
```

```
Dprintf("VideoCurrentList\n");
       while (video! = NULL) {
              if (count = = 300) Dprintf("VideoYUVList: static size exceeded\n");
              if (video->type==YUV) videoList[count++]=video->name:
              video = video - > next;
       }
       videoList(count) = NULL;
       SortList(videoList,count);
       return(videoList);
}
String *VideoDropList()
{
       static String videoList[300];
       Video video = global-> videos;
              count=0;
       int
       Boolean
                     VideoHasFrame();
       Dprintf("VideoDropList\n");
       while (video! = NULL) {
              if (False = = VideoHasFrame(video,global-> frames)) {
                     videoList[count] = video- > name;
                     count + = 1;
             };
              video = video -> next;
       }
       videoList[count] = NULL;
       SortList(videoList,count);
       return(videoList);
}
```

```
VideoHasFrame(video,frame)
Boolean
Video video;
Frame frame;
{
       if (frame = = NULL) return(False);
      else if (frame-> video = = video) return(True);
             else return(VideoHasFrame(video,frame->next));
}
       VideoLoad(w,closure,call_data)
void
Widget
              w;
              closure, call data;
caddr_t
{
       Video vid=(Video)MALLOC(sizeof(VideoRec));
       XawListReturnStruct *name = (XawListReturnStruct *)call_data;
              frame, channel;
       int
       Dprintf("VideoLoad %s\n",name-> string);
       strcpy(vid->name,name->string);
       strcpy(vid-> files, name-> string);
       vid->next=global->videos;
       global-> videos=vid;
       vid-> rate = 30;
       Parse(VID DIR, name-> string, VID_EXT);
       for (channel=0; channel < (vid-> type==MONO?1:3); channel++)
              vid->data[channel]=(short **)MALLOC(sizeof(short *)*vid->size[2]);
       if (!vid->disk) for(frame=0;frame < vid->size[2];frame++)
 GetFrame(vid, frame);
```

```
Dprintf("VideoLoad terminated\n");
      if (global->batch = = NULL) InitFrame(w.closure,call_data);
}
      VideoSave(w,closure,call_data)
void
Widget
              w;
              closure, call_data;
caddr t
{
       Video video;
       XawListReturnStruct *name=(XawListReturnStruct *)call_data;
              frame;
       int
       video = FindVideo(name-> string, global-> videos);
       if (video-> files[0] = = '\0') strcpy(video-> files, name-> string);
        SaveHeader(video);
        for (frame = 0; frame < video-> size[2]; frame + +) {
                             disk=video->disk;
               Boolean
               GetFrame(video,frame);
               video->disk=True;
               SaveFrame(video, frame);
               video->disk=disk;
               FreeFrame(video, frame);
        }
        Dprintf("VideoSave terminated\n");
 }
        VideoDTSave(w,closure,call_data)
 void
 Widget
```

```
closure. call_data;
caddr_t
{
                         Video video;
                         FILE *fp, *fopen();
                         XawListReturnStruct *name = (XawListReturnStruct *)call_data;
                                                  file_name[STRLEN], whole_frame[512][512];
                                                  frame, i, x, y, offset[2];
                          int
                          DTheader
 header = \{ \text{"DT-IMAGE"}, 1, 4, 1, 2, \text{""}, \text{""}, 1, \{0, 0, 4, 0\}, 1, 1, 0, 1, \{4, 3\}, 8, 1, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2\}, \{0, 2
  ,2},"","xwave generated image",""};
                          Dprintf("VideoDTSave %s\n",name->string);
                          video=FindVideo(name->string,global->videos);
   sprintf(file_name, "%s%s/%s/%s/%s/%s\0", global->home, IMAGE_DIR, video->path, video-
    > files, ".img");
                           offset[0] = (512\text{-video} > \text{size}[0])/2;
                            offset[1] = (512\text{-video} > \text{size}[1])/2;
                           offset[0] = offset[0] < 0?0:offset[0];
                            offset[1] = offset[1] < 0.0:offset[1];
                            fp=fopen(file name, "w");
                             fwrite(&header,1,sizeof(DTheader),fp);
                             GetFrame(video,0);
                             for(y=0;y<512;y++) for(x=0;x<512;x++) 
                                                                              X, Y, oct;
                                                     int
                                                      if (y < offset[1] \mid | x < offset[0] \mid | y - offset[1] > = video - size[1] \mid |
     x-offset[0] > = video- > size[0]) whole_frame[y][x] = 0;
                                                      else {
                                                                               if (video-> trans.type = = TRANS_Wave) {
```

```
CvtIndex(x-offset[0],y-offset[1],video->size[0],video->size[1],video->trans.wavelet.spa
ce[0],&X,&Y,&oct);
whole frame[y][x] = 128 + \text{Round}(\text{video-} > \text{data}[0][0][\text{Y*video-} > \text{size}[0] + \text{X}]*(\text{oct} = = \text{video})
-> trans. wavelet.space[0]?1:4), video-> precision);
                        } else {
                                X = x-offset[0]; Y = y-offset[1];
whole frame[y][x] = 128 + \text{Round}(\text{video} > \text{data}[0][0][Y*\text{video} > \text{size}[0] + X], \text{video} > \text{preci}
sion);
                }
        FreeFrame(video,0);
        fwrite(whole_frame,1,512*512,fp);
        fclose(fp);
}
        VideoXimSave(w,closure,call data)
void
Widget
                w;
caddr t
               closure, call data;
{
        Video video;
        FILE *fp, *fopen();
        XawListReturnStruct *name = (XawListReturnStruct *)call_data;
               file_name[STRLEN], *whole_frame;
        char
               frame, channel, i, x, y;
        int
        ImageHeader header;
        Dprintf("VideoXimSave %s\n",name->string);
```

```
video = FindVideo(name- > string, global- > videos);
       whole frame = (char *)MALLOC(video-> size[0]*video-> size[1]);
       if (video- > files[0] = = '\0') strcpy(video- > files, name- > string);
sprintf(file name, "%s%s/%s/%s%s\0", global->home, IMAGE_DIR, video->path, video-
> files, ".xim");
       fp = fopen(file name, "w");
       sprintf(header.file version, "%8d", IMAGE_VERSION);
       sprintf(header.header size, "%8d", 1024);
       sprintf(header.image width, "%8d", video-> size[0]);
       sprintf(header.image_height, "%8d", video-> size[1]);
       sprintf(header.num_colors, "%8d",256);
       sprintf(header.num_channels, "%8d", video->type==MONO?1:3);
       sprintf(header.num_pictures, "%8d", video-> size[2]);
       sprintf(header.alpha channel, "%4d",0);
       sprintf(header.runlength, "%4d",0);
       sprintf(header.author, "%48s", "xwave");
       sprintf(header.date, "%32s", "Now");
       sprintf(header.program, "%16s", "xwave");
       for(i=0;i<256;i++)
              header.c_map[i][0] = (unsigned char)i;
              header.c_map[i][1]=(unsigned char)i;
              header.c map[i][2]=(unsigned char)i;
       fwrite(&header,1,sizeof(ImageHeader),fp);
       for (frame = video- > start; frame < video- > start + video- > size[2]; frame + +) {
              GetFrame(video, frame-video- > start);
              for(channel = 0; channel < (video - > type = = MONO?1:3); channel + +) {
                     for(x=0; x < video- > size[0]; x + +)
                            for(y=0;y < video-> size[1];y++)
```

```
ss(video.channel.x,y)] > video- > precision);
                     fwrite(whole frame, sizeof(char), video- > size[0] *video- > size[1], fp);
              FreeFrame(video, frame-video-> start);
       }
       fclose(fp);
       XtFree(whole frame);
}
       VideoMacSave(w,closure,call_data)
void
Widget
caddr t
              closure, call_data;
{
       Video video;
       FILE *fp, *fopen();
       XawListReturnStruct *name = (XawListReturnStruct *)call_data;
       char
              file_name[STRLEN], *whole_frame;
       int
              frame, channel, i, x, y;
       Dprintf("VideoMacSave %s\n",name-> string);
       video=FindVideo(name-> string, global-> videos);
       if (video-> files[0] = = '\0') strcpy(video-> files,name-> string);
sprintf(file_name, "%s%s/%s/%s/%s%s\0",global->home,IMAGE_DIR,video->path,video-
> files, ".mac");
       fp = fopen(file name, "w");
      whole_frame = (char *)MALLOC(video-> size[1]*video-> size[0]*3);
       for(frame = 0; frame < video - > size[2]; frame + +) {
                     size = video - > size[0] + video - > size[1];
              int
```

```
GetFrame(video,frame);
                                            for(channel = 0; channel < (video - > type = = MONO?1:3); channel + +)
                                                                     for(x=0;x < video -> size[0];x++)
                                                                                            for(y=0; y < video -> size[1]; y++)
whole_frame[(x+video-> size \{0\} + y) + 3 + channel] = itc(video-> data[channel][frame][Addregreen] + (video-> size \{0\} + y) + (video-> data[channel][frame][Addregreen] + (video-> data[channel][frame][Addregreen][Addregreen] + (video-> data[channel][frame][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][Addregreen][
ss(video,channel,x,y)] > video-> precision);
                                              fwrite(whole frame, 1, 3*size, fp);
                                              FreeFrame(video, frame);
                        fclose(fp);
                        XtFree(whole_frame);
 }
                         VideoHexSave(w,closure,call_data)
  void
  Widget
                                                w;
                                               closure, call_data;
  caddr t
                          Video video;
                          FILE *fp, *fopen();
                          XawListReturnStruct *name=(XawListReturnStruct *)call_data;
                                                 file name[STRLEN];
                          char
                                                 frame, channel, i;
                          int
                          Dprintf("VideoHexSave %s\n",name->string);
                           video = FindVideo(name-> string, global-> videos);
                           if (video-> files[0] = = '\0') strcpy(video-> files, name-> string);
     sprintf(file_name, "%s%s/%s/%s%s\0",global->home,IMAGE_DIR,video->path,video-
       > files, ".h");
```

```
fp=fopen(file_name, "w");
      for(frame = 0; frame < (video - > size[2] > 2?2: video - > size[2]); frame + +) {
                           size = video- > size[1]*video- > size[0];
                    GetFrame(video, frame);
                    fprintf(fp, "char
%s\%d[\%d] = {\n",name-> string[strlen(name-> string)-1] = "d"?"src":"dst",frame.size);}
                    for(i=0;i < size;i++)
fprintf(fp, "0x\%02x, \%c", (video->data[0][frame][i] >> video->precision) + 128, i\%20 = = 0
19?'\n':'');
                     fprintf(fp, "\n\;\n");
                     FreeFrame(video, frame);
       }
       fclose(fp);
}
#define AB_WIDTH 1440
#define AB_HEIGHT 486
       VideoAbekusSave(w,closure,call_data)
void
 Widget
              closure, call_data;
 caddr t
 {
        AbekusCtrl ctrl=(AbekusCtrl)closure;
        FILE *fp, *fopen();
        char file_name[STRLEN], *data=(char
 *)MALLOC(AB_WIDTH*AB_HEIGHT), zero=itc(0);
               frame, channel, i, x, y, length=0;
        Video vids[4];
```

```
Dprintf("VideoAbekusSave\n");
      for(i=0;i<4;i++)
             if (ctrl->names[i]!=NULL) {
                    vids[i] = FindVideo(ctrl-> names[i], global-> videos);
                    length = length > vids[i] - > size[2]?length:vids[i] - > size[2];
             } else vids[i] = NULL;
      for(frame = 0; frame < length; frame + +) {
             sprintf(file name, "%d.yuv\0", frame + 1);
             fp=fopen(file name, "w");
             for(i=0;i<4;i++) GetFrame(vids[i],frame);
             for(y=0;y < AB_HEIGHT;y++)
                    for(x=0;x < AB WIDTH;x++) {
i=(x < AB WIDTH/2?0:1)+(y < AB HEIGHT/2?0:2),
                                        Y = y < AB HEIGHT/2?y:y-AB_HEIGHT/2,
                                        X = (x < AB WIDTH/2?x:x-AB WIDTH/2)/2,
                                        channel = ((x\&1) = 1)?0:((X\&1) = 0)?1:2;
                           if (vids[i]->type==MONO && channel!=0 | |
X > = vids[i] - size[0] \mid | Y > = vids[i] - size[1]) data[x+y*AB_WIDTH] = zero;
                           else
data[x+y*AB WIDTH] = itc(vids[i]-> data[channel][frame][Address(vids[i], channel, X, Y)]
> > vids[i]-> precision);
             for(i=0;i<4;i++) {
                    FreeFrame(vids[i],frame);
                    EraseFrame(vids[i], frame);
             fwrite(data,1,AB_WIDTH*AB_HEIGHT,fp);
             fclose(fp);
}
```

```
VideoDrop(w,closure,call_data)
void
Widget
              W;
             closure, call data;
caddr t
{
      Video *videos = &global -> videos, video;
       XawListReturnStruct *name=(XawListReturnStruct *)call_data;
              channel, frame;
       int
       Dprintf("VideoDrop %s\n",name->string);
       video=FindVideo(name->string,global->videos);
       while (*videos! = video && *videos! = NULL) videos = &((*videos)-> next);
       if (*videos!=NULL) {
              *videos = (*videos)-> next;
              for(channel=0; channel < (video-> type = = MONO?1:3); channel + +)
                     if (video- > data[channel]! = NULL) {
                            for(frame = 0; frame < video - > size[2]; frame + +)
                                   if (video->data[channel][frame]!=NULL)
XtFree(video->data[channel][frame]);
                            XtFree(video->data[channel]);
              XtFree(video);
       }
}
/* Obsolete
       VideoDiff(w,closure,call_data)
void
Widget
              w;
caddr t
              closure, call data;
.{.
```

```
XawListReturnStruct *name = (XawListReturnStruct *)call data;
       Video src=FindVideo(name-> string, global-> videos), dst=CopyHeader(src);
              frame, channel, i;
       int
       printf("VideoDiff %s\n",name->string);
       sprintf(dst->name, "%s.dif(0", src->name);
       for(frame = 0; frame < src-> size[2]; frame + +) {
              GetFrame(src,frame);
              NewFrame(dst, frame);
              for(channel = 0; channel < (video-> type = = MONO?1:3); channel + +)
                     for(i=0; i < src -> size[1]*src -> size[0]; i++)
dst->data[channel][frame][i] = src->data[channel][frame][i])-(frame = = 0?0:src->data[ch
annel][frame-1][i]);
              SaveFrame(dst, frame);
              FreeFrame(dst, frame);
              if (frame > 0) FreeFrame(src, frame-1);
       }
       FreeFrame(dst,src-> size[2]-1);
       dst-> next = global-> videos;
       global-> videos = dst;
}
*/
       VideoClean(w,closure,call_data)
void
Widget
caddr_t
             closure, call data;
{
       Video *videos=&global->videos, video;
       int
              channel, frame;
```

```
Dprintf("VideoClean\n");
      while(*videos! = NULL) {
              video = *videos;
              if (False = = VideoHasFrame(video,global-> frames)) {
                     Dprintf("Erasing video: %s\n",video->name);
for(channel = 0; channel < (video - > type = = MONO?1:3); channel + +)
                            if (video->data[channel]!=NULL) {
                                   for(frame = 0; frame < video- > size[2]; frame + +)
                                          if (video->data[channel][frame]!=NULL)
XtFree(video->data[channel][frame]);
                                   XtFree(video->data[channel]);
                            }
                     *videos = video- > next;
                     XtFree(video);
              } else videos = &(*videos)-> next;
       }
}
typedef
             struct {
      Frame frame;
      XtIntervalld id;
      unsigned long
                           interval;
      long msec, shown, average;
      Pixmap
                     *movie;
      int
             fno, old_fno;
} MovieArgRec, *MovieArg;
void
      Projector(client_data,id)
XtPointer
             client_data;
XtIntervalld
             *id;
```

```
{
                     MovieArg
                                                                 movieArg = (MovieArg)client_data;
                                                                  *dpy = XtDisplay(global- > toplevel);
                     Display
                     struct timeval
                     struct timezone
                                                                                       tzp;
                                           new_msec;
                     long
                                           scrn=XDefaultScreen(dpy);
                     int
movieArg->id=XtAppAddTimeOut(global->app_con,movieArg->interval,Projector,mo
vieArg);
                      gettimeofday(&tp,&tzp);
                      new_msec=tp.tv_sec*1000+tp.tv_usec/1000;
                     if (movieArg->msec!=0) {
movieArg->average=(movieArg->average*movieArg->shown+new_msec-movieArg-
 > msec)/(movieArg-> shown+1);
                                           movieArg->shown++;
                     movieArg->msec=new msec;
XCopyArea(dpy,movieArg->movie[movieArg->fno],XtWindow(movieArg->frame->i
mage_widget), DefaultGC(dpy,scrn),0,0,movieArg->frame->video->size[0],movieArg-
 > frame-> video-> size[1],0,0);
movieArg -> fno = movieArg -> fno = movieArg -> frame -> video -> size[2]-1?0:movieArg -> fno = movieArg -> fno = movi
g \rightarrow fno + 1;
                     StopMovie(w,closure,call_data)
void
 Widget
                                            w;
```

```
closure, call data;
caddr t
{
                   movieArg = (MovieArg)closure;
      MovieArg
                    *dpy = XtDisplay(global-> toplevel);
      Display
      int
             args[1];
      Arg
      XtRemoveTimeOut(movieArg->id);
      Dprintf("Movie showed %d frames at an average of %f
fps\n",movieArg-> shown,1000.0/(float)movieArg-> average);
       for(i=0;i < movieArg-> frame-> video-> size[2];i++)
XFreePixmap(dpy,movieArg->movie[i]);
      XtFree(movieArg-> movie);
      XtSetArg(args[0], XtNbitmap, UpdateImage(movieArg-> frame));
      XtSetValues(movieArg-> frame-> image_widget, args, ONE);
      XSynchronize(dpy,False);
} .
#define
             MOVIE ICONS
     Movie(w,closure,call_data)
Widget
             w;
             closure, call_data;
caddr t
{
      Video video=((Frame)closure)->video;
                    movieArg = (MovieArg)MALLOC(sizeof(MovieArgRec));
      MovieArg
                   shell = ShellWidget("movie", XtParent(w), SW_over, NULL, NULL),
      Widget
                    form = FormatWidget("movie_form", shell),
widgets[MOVIE_ICONS];
```

```
*dpy = XtDisplay(globai- > toplevel);
Display
             items[] = {
Formltem
      {"movie_stop", "stop", 0, 0, FW_icon, NULL},
};
                    callbacks[] = {
XtCallbackRec
      {StopMovie.(caddr_t)movieArg},
      {Free,(caddr_t)movieArg},
       {Destroy,(caddr_t)shell},
       {NULL, NULL},
};
       i;
int
XGCValues values;
GC
       gc;
Dprintf("Movie\n");
FillForm(form, MOVIE\_ICONS, items, widgets, callbacks);
 XtPopup(shell, XtGrabExclusive);
 values.foreground=255;
 values.background = 0;
 gc = XtGetGC(XtParent(w),GCForeground \mid GCBackground, \&values);\\
 movieArg-> frame=(Frame)closure;
 movieArg->movie=(Pixmap *)MALLOC(video->size[2]*sizeof(Pixmap));
 movieArg->old_fno=movieArg-> frame-> frame;
 for(i=0; i < video-> size[2]; i++) 
              fno[STRLEN];
        char
        sprintf(fno, "%03d\0", i+video-> start);
        movieArg-> frame-> frame=i;
        GetFrame(video,i);
        movieArg->movie[i] = UpdateImage(movieArg-> frame);
```

```
XDrawImageString(dpy,movieArg->movie[i],gc,video->size[0]-50,10.fno,3);
XCopyArea(dpy,movieArg->movie[i],XtWindow(movieArg->frame->image_widget),D
efaultGC(dpy,0),0,0,video->size[0],video->size[1],0,0);
                                          movieArg-> frame-> frame = movieArg-> old_fno;
                                          FreeFrame(video,i);
                     }
                     XtDestroyGC(gc);
                     movieArg -> fno = 0;
                     movieArg-> msec=0;
                      movieArg-> shown=0;
                      movieArg->average=0;
                      movieArg->interval=1000/video->rate;
movie Arg->id=XtAppAddTimeOut(global->app\_con, movie Arg->interval, Projector, movie Arg->in
 vieArg);
                      XSynchronize(dpy, True);
 } .
                      Compare(w,closure,cail_data)
 void
  Widget
                                            closure, call_data;
 caddr t
  {
                      XawListReturnStruct *name = (XawListReturnStruct *) call\_data;
                       Video src=(Video)closure, dst=FindVideo(name->string,global->videos);
                                            channels=src->type==MONO || dst->type==MONO?1:3, channel,
                       int
  values = 0, x, y,
                                                                  frames = src - size[2] > dst - size[2]?dst - size[2]:src - size[2],
   frame;
```

```
double
                                                         mse;
                                                         msg = NewMessage(NULL, 400);
                  Message
                  XtCallbackRec
                                                                            callbacks[] = {
                                      {CloseMessage,(caddr_t)msg}, {NULL,NULL},
                  };
                  msg->rows=frames>5?10:2*frames; msg->cols=40;
                   if (global->batch==NULL)
MessageWindow(FindWidget("frm_compare", w), msg, "Compare", True, callbacks);
                   for(frame = 0; frame < frames; frame + +) {
                                                                             srcp=src->precision>dst->precision;
                                       Boolean
                                                          err_sqr=0,
                                       int
precision=srcp?src->precision-dst->precision:dst->precision-src->precision;
                                       Mprintf(msg, "Compare: %s%03d and
 %s\%03d\n",src->name,src->start+frame,dst->name,dst->start+frame);
                                       GetFrame(src,frame);
                                       GetFrame(dst, frame);
                                       for(channel = 0; channel < channels; channel + +) {
 values + = Size(src-> size[1] > dst-> size[1]?dst:src,channel,1)*Size(src-> size[0] > dst-> s
 ize[0]?dst:src,channel,0);
 for(y=0;y < Size(src-> size[1] > dst-> size[1]?dst:src,channel,1);y++)
  for(x=0;x < Size(src-> size[0] > dst-> size[0]?dst:src,channel,0);x++) 
  err = (src -  data[channel][frame][x + Size(src, channel, 0)*y] <  (srcp?0:precision))-(dst->
  \label{lem:data} $$  \data[channel][frame][x+Size(dst,channel,0)*y] < (srcp?precision:0)); $$  \data[channel][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame][frame
                                                                                                   err sqr + = err*err;
                                                                                }
```

```
- 390 -
```

```
FreeFrame(src,frame);
          FreeFrame(dst,frame);
          mse = (double)err_sqr/(double)(values);
          Mprintf(msg, "Error %d MSE %f PSNR
ion)))-1),2.0)/mse));
           Mflush(msg);
     }
}
     BatchCompare(w,closure,call_data)
void
Widget
caddr_t
           closure, call_data;
{
     String name = (String) closure;
     closure = (caddr_t)FindVideo(name, global-> videos);
     Compare(w,closure,call_data);
}
```

source/xwave.c

```
"../include/xwave.h"
#include
             <X11/Xresource.h>
#include
             <X11/Intrinsic.h>
#include
             <X11/Quarks.h>
#include
                    ReOrderPalettes();
extern Palette
extern void
             NameButton();
            ImageNotify();
extern void
extern void
           Parse();
#define
             IconPath
                           "bitmaps"
#define
             IconFile
                           "xwave.icons"
#define
             CompressPath
             CompressExt ".compress"
#define
#define
             PalettePath
#define
             PaletteExt
                           ".pal"
Global
             global;
String ChannelName[3][4]={
       {"GreyScale", NULL, NULL, NULL},
       {"Red ","Green","Blue ","Color"},
       {"Y-Lumunance", "U-Chrome ", "V-Chrome
                                                                 "},
                                                   ","Color
};
             XtNdebug "debug"
#define
              XtNbatch "batch"
#define -
```

```
static XtResource
                    resources[] = {
      {XtNdebug, XtCBoolean, XtRBoolean, sizeof(Boolean),
      XtOffset(Global, debug), XtRString, "false"},
      {X:Nbatch, XtCFile, XtRString, sizeof(String),
      XtOffset(Global, batch), XtRString, NULL},
};
static XrmOptionDescRec options[]={
      {"-debug", "*debug", XrmoptionNoArg, "true"},
       {"-batch", "*batch", XrmoptionSepArg, NULL},
};
static Boolean
                    CvtStringToPixel2();
#if defined( STDC )
                                 const colorConvertArgs[2];
externalref
             XtConvertArgRec
#else
externalref XtConvertArgRec colorConvertArgs[2];
#endif
static String fallback resources[]={
       "*copy video*Toggle*translations: #override \\n < Btn1Down>, < Btn1Up>:
set() notify()",
       "*copy video*copy*state: true",
       NULL.
};
XtActionsRec
                    actionTable[] = {
       {"NameButton", NameButton},
};
main(argc,argv,envp)
```

```
int
       argc;
       *argv[], *envp[];
char
{
             InitPixmaps(), InitActions(), InitMain(), InitEnv(), InitDither(), Dispatch();
       GlobalRec
                    globalrec;
       global=&globalrec;
       global-> videos = NULL;
       global-> frames = NULL;
       global->points=NULL;
       InitEnv(envp);
global->toplevel=XtAppInitialize(&(global->app_con), "xwave", options, XtNumber(optio
ns),&argc,argv,fallback_resources,NULL,ZERO);
XtGetApplicationResources(global->toplevel,global,resources,XtNumber(resources),NUL
L,ZERO);
      if (global->batch!=NULL) {
             Parse(BATCH DIR, global->batch, BATCH EXT);
             if (global->batch list!=NULL) Dispatch(global->batch list);
      }
      if (global->batch = = NULL) {
             XtAppAddActions(global->app_con,actionTable,XtNumber(actionTable));
XtSetTypeConverter(XtRString,XtRPixel,CvtStringToPixel2,colorConvertArgs,XtNumber
(colorConvertArgs), XtCacheByDisplay, NULL);
             if (global->debug) Dprintf("Xwave Debugging Output\n");
             InitVisual();
             InitDither();
             InitPixmaps(IconPath, IconFile);
             Parse(PalettePath, "xwave", PaletteExt);
```

```
giobal-> palettes = ReOrderPalettes(global-> palettes, global-> palettes);
             InitActions(global->app con);
             InitMain();
             XtRealizeWidget(global->toplevel);
             XtAppMainLoop(global->app_con);
      }
}
void
      InitEnv(envp)
char
       *envp[];
{
       String home=NULL, xwave=NULL;
       Dprintf("Initializing environment\n");
       while(*envp!=NULL) {
             if(!strncmp(*envp, "HOME=",5)) home = (*envp) +5;
             if(!strncmp(*envp, "XWAVE=",6)) xwave=(*envp)+6;
             envp++;
       if (xwave!=NULL) sprintf(global->home, "%s/", xwave);
       else sprintf(global->home, "%s/xwave/",home);
}
#define
             HEIGHT
                           14
       InitPixmaps(path, file)
void
char
       *file, *path;
```

```
FILE *fp, *fopen();
             icons;
      Icon
            pad[100];
      char
                    *dpy = XtDisplay(global-> toplevel);
      Display
             i, j, sink, scrn=XDefaultScreen(dpy), depth=DisplayPlanes(dpy,scrn),
      int
                    bpl = (global - > levels * depth + 7)/8;
             data[HEIGHT*bpl];
      char
      XImage
*image=XCreateImage(dpy,global->visinfo->visual,depth,ZPixmap,0,data,global->leve
ls, HEIGHT, 8, bpl);
       sprintf(pad, "%s%s/%s\0", global-> home, path, file);
       if (NULL == (fp = fopen(pad, "r"))) 
             Eprintf("Can't open file %s\n",pad);
             exit();
       }
       fscanf(fp, "%d\n",&global->no_icons);
       global->icons=(Icon)MALLOC((1+global->no_icons)*sizeof(IconRec));
       for(i=0;i < global > no_icons;i++) {
              global->icons[i].name=(String)MALLOC(100);
              fscanf(fp, "%s\n", global-> icons[i].name);
              sprintf(pad, "%s%s/%s\0",global->home,path,global->icons[i].name);
              XReadBitmapFile(
                     XtDisplay(global->toplevel),
                     XDefaultRootWindow(dpy),
                     pad,
                     &global->icons[i].width,
                     &global->icons[i].height,
                     &global->icons[i].pixmap,
                     &sink,
                     &sink
              );
```

```
}
                       global->icons[global->no_icons].name = (String)MALLOC(100);
                       strcpy(global->icons[global->no_icons].name, "colors");
                       global->icons(global->no_icons).width=global->levels;
                       global->icons[global->no_icons].height=HEIGHT;
                       for(i=0; i < g!obal-> levels; i++)
                                             for(j=0; j < HEIGHT; j++) XPutPixel(image, i, j, i);
global->icons[global->no\_icons].pixmap=XCreatePixmap(dpy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(dpsy,XDefaultRootWindow(
y), global-> levels, HEIGHT, depth);
XPutImage(dpy,global->icons[global->no_icons].pixmap,DefaultGC(dpy,scrn),image,0,0
 ,0,0,global->levels,HEIGHT);
                       global->no_icons++;
                      XtFree(image);
                       fclose(fp);
#define done(type, value) \
                       {\
                                             if (toVal->addr!= NULL) {
                                             if (toVal-> size < sizeof(type)) {
                                                                    toVal-> size = sizeof(type);
                                                                    return False;
                                             }\
                                              (type^*)(toVal->addr) = (value);
                                             }
                                             else {
                                             static type static_val;
                                             static_val = (value);
                                             toVal->addr = (XtPointer)&static_val;
```

```
toVal-> size = sizeof(type);
             return True;
      }
              dist(colora,colorb) \
#define
abs(colora.red-colorb.red) + abs(colora.green-colorb.green) + abs(colora.blue-colorb.blue)
static Boolean CvtStringToPixel2(dpy, args, num_args, fromVal, toVal, closure_ret)
   Display* dpy;
   XrmValuePtr args;
   Cardinal
               *num_args;
   XrmValuePtr
                     fromVal;
                     toVal:
   Xrm Value Ptr
   XtPointer *closure_ret;
{
                 str = (String)fromVal->addr;
   String
   XColor
                 screenColor;
   XColor
                 exactColor;
   Screen
                 *screen;
   Colormap
                 colormap;
   Status
                 status;
   String
                 params[1];
   Cardinal
                 num_params=1;
       Dprintf("Convert string to pixel 2\n");
   if (*num_args != 2)
    XtAppErrorMsg(XtDisplayToApplicationContext(dpy), \ "wrongParameters",\\
 "cvtStringToPixel",
                 "XtToolkitError",
        "String to pixel conversion needs screen and colormap arguments",
       (String *)NULL, (Cardinal *)NULL);
```

```
screen = *((Screen **) args[0].addr);
  colormap = *((Colormap *) args[1].addr);
      if (!strcmp(str,XtDefaultBackground)) {
             *closure_ret = False;
             done(Pixel, WhitePixelOfScreen(screen));
      }
      if (!strcmp(str,XtDefaultForeground)) {
              *closure ret = False;
             done(Pixel, BlackPixelOfScreen(screen));
      }
      params[0] = str;
      if (0 = XParseColor(DisplayOfScreen(screen), colormap, str, \&screenColor)) \  \{\\
              XtAppWarningMsg(XtDisplayToApplicationContext(dpy), "noColormap",
"cvtStringToPixel",
                     "XtToolkitError", "Cannot parse color: \"%s\"",
params, & num_params);
              return False;
      } else {
      if (0 = = XAllocColor(DisplayOfScreen(screen),colormap,&screenColor)) {
                     int
                            i, delta, closest=0;
                                   colors[global->levels];
                     XColor
                     for(i=0;i < global-> levels;i++) colors[i].pixel=i;
XQueryColors(DisplayOfScreen(screen),colormap,colors,global->levels);
                     delta = dist(screenColor,colors[0]);
                     for(i=1;i < global > levels;i++) {
                                   delta new=dist(screenColor,colors[i]);
                            int
                            if (delta_new < delta) {
                                   delta = delta new;
```

```
- 399 -
```

```
closest = i:
                              }
                      Dprintf("Closest color to %s is pixel %d red %d green %d blue
 \%d\n".str.colors[closest].pixel,colors[closest].red,colors[closest].green,colors[closest].blue
);
                       *closure ret = (char*)True;
                       done(Pixel, closest);
               } else {
                       *closure ret = (char*)True;
                       done(Pixel, screenColor.pixel);
               }
 }
        Dispatch(list)
 void
 Batch list;
. {
        if (list-> next! = NULL) Dispatch(list-> next);
         (list->proc)(NULL, list->closure, list->call_data);
         if (list->closure! = NULL) XtFree(list->closure);
         if (list->call_data!=NULL) XtFree(list->call_data);
         XtFree(list);
  }
         BatchCtrl(w,closure,call_data)
  void
  Widget
                w;
                closure, call_data;
  caddr_t
```

```
{
        Dprintf("BatchCtrl\n");
        global->batch=(String)closure;
 }
        UnixShell(w,closure,call_data)
 void
 Widget
               closure, call_data;
 caddr t
  {
        if (-1 = Fork((char **)closure)) Eprintf("Unable to fork\n");
  }
        InitDither()
  void
  {
               i, j, k, l,
         int
                      dm4[4][4] = {
                             0, 8, 2, 10,
                             12, 4, 14, 6,
                             3, 11, 1, 9,
                             15, 7, 13, 5
                      };
         for(i=0;i<4;i++)
                for(j=0; j<4; j++)
                      for(k=0;k<4;k++)
                             for(1=0;1<4;1++)
   global->dither[4*k+i][4*l+j]=(dm4[i][j]<<4)+dm4[k][l];
..}
```

- 401 -

source/Copy.h

```
typedef struct {
    Video video;
    char name[STRLEN], src_name[STRLEN];
    int UVsample[2];
    int mode;
    Widget radioGroup;
} CopyCtrlRec, *CopyCtrl;
```

```
source/Gram.y
```

```
%{
/*
      Grammar for files: .elo
 */
              "../include/xwave.h"
#include
              "Klics.h"
#include
              "Transform.h"
#include
              "Copy.h"
#include
              "Video.h"
#include
              VideoLoad();
extern void
              VideoSave();
extern void
              VideoDrop();
extern void
extern void
              ImportKlics();
              VideoAbekusSave();
extern void
              UnixShell();
extern void
              BatchCompCtrl();
extern void
              BatchTransCtrl();
 extern void
              BatchCopyCtrl();
 extern void
              BatchCompare();
 extern void
              BatchCtrl();
 extern void
 extern CompCtrl
                     InitCompCtrl();
 extern CopyCtrl
                     InitCopyCtrl();
                      InitTransCtrl();
 extern TransCtrl
 static char
               *ptr;
 void
        NewBatch();
 %}
```

```
%union
{
                fnum;
     double
     int
          num;
          *ptr;
     char
                bool;
     Boolean
};
          SIZE TRANSFORM TRANSFORM_NONE TRANSFORM_WAVE PATH
%token
          FILE PAL PALETTE RANGE LINE
%token
          FILE_VID TYPE FORMAT_MONO FORMAT_RGB FORMAT_YUV
%token
RATE DISK GAMMA PATH FILES START END LEN DIM HEADER OFFSETS
NEGATIVE PRECISION
          FILE_BAT LOAD SAVE SAVE_ABEKUS COMPARE DROP
%token
COMPRESS VIDEO_NAME STATS_NAME BIN_NAME
           STILL_MODE VIDEO_MODE AUTO_Q QUANT_CONST
%token
THRESH_CONST BASE_FACTOR DIAG_FACTOR CHROME_FACTOR
           DECISION DEC_MAX DEC_SIGABS DEC_SIGSQR FEEDBACK
%token
FILTER FLT_NONE FLT_EXP CMP_CONST SPACE LEFT_BRACE RIGHT_BRACE
DIRECTION
           FPS BITRATE BUFFER XWAVE SHELL IMPORT_KLICS
 %token
           COPY DIRECT_COPY DIFF LPF_WIPE LPF_ONLY RGB_YUV
 %token
                      NUMBER
           < mm >
 %token
                      STRING
           <ptr>
 %token
                      FNUMBER
            <fnum>
 %token
                      BOOLEAN
            <bool>
 %token
                 number video_type decision filter
 %type < num >
                 string
 %type <ptr>
                 fnumber
 %type < frum >
                 boolean
 %type <bool>
```

```
%start wait
%%
wait
             | pal_id pal_desc
             | video_id video_desc
             | bat_id bat_desc bat_end;
pal_id : FILE_PAL {
                    Dprintf("Gram: palette file %s\n",global->parse_file);
             };
              : FILE_VID {
video_id
                           Dprintf("Gram: video file %s\n",global->parse_file);
                           global-> videos-> start=1;
                            global->videos->size[2]=1;
                     };
              : FILE_BAT {
 bat_id
                            Dprintf("Gram: batch file %s\n",global->parse_file);
                     };
 pal_desc
                     | pal_desc palette LEFT_BRACE mappings RIGHT_BRACE;
                     : PALETTE string {
 palette
                                          pal = (Palette)MALLOC(sizeof(PaletteRec));
                            Palette
                            Dprintf("Gram: palette %s\n",$2);
                            strcpy(pal->name,$2);
                            pal->mappings=NULL;
```

```
- 405 -
```

```
pal-> next = global-> palettes;
                          global->palettes=pal;
                          global->no_pals++;
                   };
mappings
                    | mappings mapping;
             : RANGE number number LINE number number {
mapping
                    Map map = (Map)MALLOC(sizeof(MapRec));
                    Dprintf("Gram: Range %d to %d m = %d c = %d n".$2,$3,$5,$6);
                    map-> start=$2;
                    map-> finish = $3;
                    map->m=$5;
                    map->c=$6;
                    map->next=global->palettes->mappings;
                    global->palettes->mappings=map;
             };
video desc
              : video_defs {
                           if (global-> videos-> size[0] = = 0 &&
 global - > videos - > size[1] = = 0) {
                                 global->videos->size[0] = global->videos->cols;
                                 global->videos->size[1]=global->videos->rows;
                           }
                    };
 video_defs
                     | video defs video_def;
              : PATH string {
 video_def
```

```
Dprintf("Video path %s\n",$2);
      strcpy(global->videos->path.$2);
| FILES string {
      Dprintf("Frames stored in %s\n",$2);
      strcpy(global-> videos-> files,$2);
| TYPE video_type {
      String types[]={"Mono","RGB","YUV"};
      Dprintf("Video type: %s\n",types[32]);
      global->videos->type=(VideoFormat)$2;
}
| RATE number {
      Dprintf("Video rate %d fps\n",$2);
       global-> videos-> rate = $2;
DISK {
       Dprintf("Frames on disk\n");
       global-> videos-> disk = True;
}
| GAMMA {
       Dprintf("Gamma corrected\n");
       global-> videos-> gamma = True;
| NEGATIVE {
       Dprintf("Negative video\n");
       global->videos->negative=True;
 | TRANSFORM video_transform
 | START number {
       Dprintf("Video start %03d\n",$2);
```

```
global-> videos-> start = $2;
| END number {
      Dprintf("Video end %03d\n",$2);
      global-> videos-> size[2] = $2-global-> videos-> start+1;
| LEN number {
      Dprintf("Video frames %d\n",$2);
      global - > videos - > size[2] = $2;
DIM number number {
       Dprintf("Video dimensions %d %d\n",$2,$3);
       global-> videos-> cols = $2;
       global-> videos-> rows = $3;
}
 | HEADER number {
       Dprintf("Video header size %d\n",$2);
       global-> videos-> offset = $2;
 | OFFSETS number number {
       Dprintf("Video offsets %d %d\n",$2,$3);
       global - > videos - > x_offset = $2;
       global->videos->y_offset=$3;
 }
 | SIZE number number {
       Dprintf("Video size %d %d\n",$2,$3);
        global->videos->size[0] =$2;
        global->videos->size[1]=$3;
 | PRECISION number {
        Dprintf("Video precision %d bits\n",8+$2);
        global->videos->precision=$2;
```

```
};
            : FORMAT_MONO { $$=(int)MONO; }
video type
                  | FORMAT_RGB { $$=(int)RGB; }
                  | FORMAT_YUV number number { $$=(int)YUV;
global->videos->UVsample[0] = $2; global->videos->UVsample[1] = $3; };
                   : TRANSFORM_NONE {
video_transform
                               global->videos->trans.type=TRANS_None;
                         | TRANSFORM_WAVE number number boolean {
                                Dprintf("Video wavelet transformed %d %d
%s\n",$2,$3,$4?"True":"False");
                                global->videos->trans.type=TRANS_Wave;
                                global->videos->trans.wavelet.space[0] = $2;
                                global->videos->trans.wavelet.space[1]=$3;
                                global-> videos-> trans.wavelet.dirn = $4;
                         };
bat_end
                    | XWAVE {
                          Dprintf("Gram: XWAVE\n");
                          NewBatch(BatchCtrl,(caddr_t)NULL,NULL);
                   };
             : bat_cmds {
 bat_desc
                          Dprintf("Gram: End of batch file\n");
                    };
 bat_cmds
                    | bat_cmds bat_cmd;
```

```
: simple_cmd
bat_cmd
                    | complex cmd
simple_cmd : LOAD string {
                          XawListReturnStruct *list_return=(XawListReturnStruct
*)MALLOC(sizeof(XawListReturnStruct));
                           Dprintf("Gram: LOAD %s\n",$2);
                           list_return->string=$2;
                           NewBatch(VideoLoad, NULL, (caddr_t) list_return);
                    }
                    | SAVE string {
                           XawListReturnStruct *list_return=(XawListReturnStruct
*)MALLOC(sizeof(XawListReturnStruct));
                           Dprintf("Gram: SAVE %s\n",$2);
                           list return->string=$2;
                           NewBatch(VideoSave, NULL, (caddr_t) list_return);
                    }
                     | SAVE_ABEKUS string string string string {
                          ... AbekusCtrl
 ctrl = (AbekusCtrl)MALLOC(sizeof(AbekusCtrlRec));
                           Dprintf("Gram: SAVE_ABEKUS %s %s %s
 %s\n",$2,$3,$4,$5);
                            strcpy(ctrl->names[0],$2);
                            strcpy(ctrl->names[1],$3);
                            strcpy(ctrl->names[2],$4);
                            strcpy(ctrl->names[3],$5);
                            NewBatch(VideoAbekusSave,(caddr\_t)ctrl,NULL);\\
                     }
```

```
| COMPARE string string {
                           XawListReturnStruct *list_return=(XawListReturnStruct
*)MALLOC(sizeof(XawListReturnStruct));
                           Dprintf("Gram: COMPARE %s with %s\n",$2,$3);
                           list_return-> string = $2;
                           NewBatch(BatchCompare,(caddr_t)$3,(caddr_t)list_return);
                     | DROP string {
                           XawListReturnStruct *list_return=(XawListReturnStruct
*)MALLOC(sizeof(XawListReturnStruct));
                           Dprintf("Gram: DROP %s\n",$2);
                           list_return-> string = $2;
                           NewBatch(VideoDrop, NULL, (caddr_t) list_return);
                     | IMPORT_KLICS string {
                           XawListReturnStruct *list_return=(XawListReturnStruct
 *)MALLOC(sizeof(XawListReturnStruct));
                            Dprintf("Gram: IMPORT_KLICS %s\n",$2);
                            list_return-> string=$2;
                            NewBatch(ImportKlics, NULL, (caddr\_t) list\_return);\\
                     | SHELL string {
                                   **argv, *str=$2;
                            char
                                   c, argc=1, len=strlen(str);
                            int
                            Dprintf("Shell %s\n",str);
                            for(c=0;c<len;c++) if (str[c]==' ') {
                                   str[c] = '\0';
                                   argc++;
```

```
- 411 -
```

```
argv = (char **)MALLOC((argc+1)*sizeof(char *));
                         argc = 0;
                         for(c=0;c < len;c+=1+strlen(str+c)) 
                                argv[argc] = (char
*)MALLOC((strlen(str+c)+1)*sizeof(char));
                                strcpy(argv[argc],str+c);
                                argc++;
                         argv[argc] = NULL;
                         NewBatch(UnixShell,(caddr_t)argv,NULL);
                   };
                   : compress LEFT_BRACE comp_args RIGHT_BRACE
complex_cmd
                   transform LEFT_BRACE trans_args RIGHT_BRACE
                   copy copy_arg;
             : COMPRESS string {
compress
                         CompCtrl
                                      ctrl = InitCompCtrl($2);
                         Dprintf("Gram: COMPRESS\n");
                         NewBatch(BatchCompCtrl,(caddr_t)ctrl,NULL);
                   };
             : TRANSFORM string {
transform
                                      ctrl=InitTransCtrl($2);
                          TransCtrl
                         Dprintf("Gram: TRANSFORM\n");
                          NewBatch(BatchTransCtrl,(caddr_t)ctrl,NULL);
                   };
             : COPY string string {
сору
```

```
ctrl=InitCopyCtrl($2);
                           CopyCtrl
                           Dprintf("Gram: Copy\n");
                           strcpy(ctrl->name,$3);
                           NewBatch(BatchCopyCtrl,(caddr\_t)ctrl,NULL);\\
                    };
comp_args
                     comp_args comp_arg;
trans args
                     trans_args trans_arg;
             : DIRECT_COPY number number {
copy_arg
                           Dprintf("Gram: Direct Copy (sample %d %d)\n",$2,$3);
                           ((CopyCtrl)global->batch_list->closure)->mode=1;
((CopyCtrl)global-> batch\_list-> closure)-> UV sample[0] = \$2;
((CopyCtrl)global-> batch_list-> closure)-> UVsample[1]=$3;
                     | DIFF {
                            Dprintf("Gram: Differance Copy\n");
                            ((CopyCtrl)global->batch_list->closure)-> mode = 2;
                     }
                     | LPF_WIPE {
                            Dprintf("Gram: LPF zero\n");
                            ((CopyCtrl)global-> batch\_list-> closure)-> mode=3;\\
                     | LPF_ONLY {
                            Dprintf("Gram: LPF only\n");
                            ((CopyCtrl)global-> batch\_list-> closure)-> mode=4;\\
                     }
```

```
| RGB_YUV {
                         Dprintf("Gram: RGB/YUV\n");
                         ((CopyCtrl)global->batch_list->closure)->mode=5;
                   | GAMMA {
                         Dprintf("Gram: Gamma convert\n");
                         ((CopyCtrl)global->batch_list->closure)->mode=6;
                   };
            : VIDEO_NAME string {
comp_arg
                         Dprintf("Gram: Compress name %s\n",$2);
strcpy((((CompCtrl)global->batch_list->closure)->name,$2);
                    | STATS_NAME string {
                          Dprintf("Gram: Stats name %s\n",$2);
strcpy(((CompCtrl)global->batch_list->closure)->stats_name,$2);
((CompCtrl)global->batch_list->closure)->stats_switch=True;
                    | BIN_NAME string {
                          Dprintf("Gram: Bin name %s\n",$2);
strcpy((((CompCtrl)global->batch_list->closure)->bin_name,$2);
((CompCtrl)global->batch_list->closure)->bin_switch=True;
                    STILL_MODE {
                          Dprintf("Gram: Still\n");
                          ((CompCtrl)global->batch_list->closure)->stillvid=True;
                    }
```

```
| VIDEO_MODE {
                         Dprintf("Gram: Video\n");
                         ((CompCtrl)global->batch_list->closure)->stillvid=False;
                   | AUTO Q boolean {
                         Dprintf("Gram: Auto_q %s\n",$2?"True":"False");
                         ((CompCtrl)global-> batch_list-> closure)-> auto_q = $2;
                   | QUANT_CONST fnumber {
                         Dprintf("Gram: Quant const %f\n",$2);
((CompCtrl)global->batch_list->closure)->quant_const=$2;
                   | THRESH_CONST fnumber {
                         Dprintf("Gram: Thresh const %f\n",$2);
((CompCtrl)global->batch_list->closure)->thresh_const=$2;
                   BASE FACTOR number fnumber {
                          Dprintf("Gram: Base factor oct \%d = \%f \ n",$2,$3);
((CompCtrl)global->batch_list->closure)->base_factors[$2]=$3;
                   | DIAG_FACTOR fnumber {
                          Dprintf("Gram: Diag factor %f\n",$2);
                          ((CompCtrl)global->batch_list->closure)->diag_factor=$2;
                   | CHROME_FACTOR fnumber {
                          Dprintf("Gram: Chrome factor %f\n",$2);
((CompCtrl)global->batch_list->closure)->chrome_factor=$2;
```

```
| DECISION decision {
                          Dprintf("Gram: Decision changed\n");
                          ((CompCtrl)global->batch_list->closure)->decide=$2;
                   | FEEDBACK number {
                          ((CompCtrl)global->batch list->closure)-> feedback=$2;
                          ((CompCtrl)global->batch_list->closure)->auto_q=True;
                   | FILTER filter {
                          String filters[2]={"None", "Exp"};
                          Dprintf("Gram: Filter %s\n",filters[$2]);
                          ((CompCtrl)global->batch list->closure)->filter=$2;
                   | CMP_CONST fnumber {
                          Dprintf("Gram: Comparison %f\n",$2);
                          ((CompCtrl)global->batch_list->closure)->cmp_const=$2;
                   | FPS fnumber {
                          Dprintf("Gram: Frame Rate %f\n",$2);
                          ((CompCtrl)global->batch_list->closure)->fps=$2;
                   | BITRATE number {
                          Dprintf("Gram: %dx64k/s\n",$2);
                          ((CompCtrl)global->batch list->closure)->bitrate=$2;
                   }
                   | BUFFER {
                          Dprintf("Gram: Buffer on\n");
((CompCtrl)global->batch_list->closure)->buf_switch=True;
                   };
             : DEC MAX{ $$ = 0; }
decision
```

filter

```
- 416 -
                   | DEC SIGABS { $$ = 1; }
                   | DEC SIGSQR { $$ = 2; };
             : FLT NONE \{ \$\$ = 0; \}
                   | FLT EXP { $$ = 1, };
            : VIDEO_NAME string {
trans_arg
                          Dprintf("Gram: Transform name %s\n",$2);
strcpy((((TransCtrl)global->batch_list->closure)->name,$2);
                    | DIRECTION boolean {
                          Dprintf("Gram: Direction %s\n",$2?"True":"False");
                          ((TransCtrl)global->batch_list->closure)->dirn=$2;
                    }
                    | SPACE number number {
                          Dprintf("Gram: Space %d %d\n",$2,$3);
                          ((TransCtrl)global->batch_list->closure)->space[0]=$2;
                           ((TransCtrl)global->batch_list->closure)->space[1]=$3;
                    }
                    | PRECISION number {
                           Dprintf("Gram: Precision %d bits\n",8+$2);
                           ((TransCtrl)global->batch_list->closure)->precision=$2;
                    };
              : BOOLEAN \{ \$\$ = \$1; \};
boolean
string: STRING
                    ptr = (char *)malloc(strlen($1)+1);
                    strcpy(ptr,1+$1);
                    ptr[strlen(ptr)-1] = '\0';
```

\$\$ = ptr;

```
};
             : FNUMBER \{ \$\$ = \$1; \};
fnumber
             : NUMBER { $$ = $1; };
%%
yyerror(s) char *s; {
      Eprintf("Gram: error %s\n",s);
      exit(3);
}
void NewBatch(proc,closure,call_data)
Proc proc;
caddr_t
             closure, call_data;
                           Batch bat = (Batch)MALLOC(sizeof(BatchRec));
                           bat->proc=proc;
                           bat-> closure = closure;
                           bat->call_data=call_data;
                           bat-> next = global-> batch_list;
                           global->batch_list=bat;
}
```

source/Klics.h

WO 94/23385

```
/* Block size - no not change */
#define
         BLOCK
typedef int Block[BLOCK][BLOCK]; /* small block */
/* tokens */
                   15
#define
         TOKENS
#define ZERO_STILL
#define NON ZERO_STILL
#define BLOCK SAME
                      2
#define ZERO VID
#define BLOCK CHANGE
#define LOCAL_ZERO
#define LOCAL_NON_ZERO
#define CHANNEL ZERO
#define CHANNEL_NON_ZERO
#define OCT_ZERO
#define OCT_NON_ZERO
                      10
#define LPF_ZERO
                    11
#define LPF_NON_ZERO
                      12
#define LPF LOC ZERO
                      13
#define LPF_LOC_NON_ZERO 14
static int
                        token_bits[TOKENS]
```

```
/* decision algorithms */
#define MAXIMUM 0
#define SIGABS 1
#define SIGSQR 2
/* compression modes */
#define STILL 0
#define SEND
#define VOID
#define STOP
/* LookAhead histogram */
                            400
#define
              HISTO
                                  20.0
#define
              HISTO_DELTA
#define
              HISTO BITS 9
              "../include/Bits.h"
 #include
              struct {
 typedef
       Video src, dst;
                     stillvid, stats switch, bin_switch, auto_q, buf_switch;
       Boolean
                     quant const, thresh_const, cmp_const, fps,
       double
                     base_factors[5], diag_factor, chrome_factor;
              bitrate, feedback, decide, filter;
        int
              name[STRLEN], stats_name[STRLEN], bin_name[STRLEN],
 src name[STRLEN];
        Bits
              bfp;
 } CompCtrlRec, *CompCtrl;
               struct {
 typedef
                     stillvid, auto_q, buf_switch;
        Boolean
                     quant_const, thresh_const, cmp_const, fps,
        double
```

- 420 -

```
base_factors[5], diag_factor, chrome_factor;
int decide;
VideoFormat type;
Boolean disk, gamma;
int rate, start, size[3], UVsample[2];
VideoTrans trans;
int precision;
} KlicsHeaderRec, *KlicsHeader;
```

source/KlicsSA.h

```
#include < stdio.h>
#include
              "Bits.h"
             negif(bool, value)
                                  ((bool)?-(value):(value))
#define
extern Bits
             bopen();
             bclose(), bread(), bwrite(), bflush();
extern void
/* Stand Alone definitions to replace VideoRec & CompCtrl assumes:
       video->type == YUV;
       video->UVsample[]={1,1};
 * video-> trans. wavelet. space [] = \{3,2\};
       ctrl->bin_switch == True;
 */
                           352
#define SA_WIDTH
#define SA_HEIGHT
                                  288
                                  2
#define
              SA_PRECISION
static double
                    base_factors[5] = \{1.0,0.32,0.16,0.16,0.16\};
#define
              diag factor
                                  1.4142136
#define chrome_factor
                           2.0
#define
              thresh const 0.6
#define
              cmp_const
                                  0.9
/* Block size - no not change */
#define
                           2
             BLOCK
typedef int Block[BLOCK][BLOCK]; /* small block */
```

```
/* tokens */
                    15
#define
          TOKENS
#define ZERO STILL
                     0
#define NON_ZERO_STILL
                      2
#define BLOCK_SAME
                     3
#define ZERO_VID
#define BLOCK CHANGE
                      5
#define LOCAL_ZERO
#define LOCAL NON_ZERO
                        6
#define CHANNEL ZERO
#define CHANNEL_NON_ZERO
#define OCT_ZERO
                       10
#define OCT_NON_ZERO
#define LPF_ZERO
                     11
                       12
#define LPF_NON_ZERO
                       13
#define LPF_LOC_ZERO
#define LPF_LOC_NON_ZERO
static int
                         token_bits[TOKENS]
/* decision algorithms */
#define MAXIMUM 0
#define SIGABS 1
#define SIGSQR 2
/* compression modes */
#define STILL 0
#define SEND
#define VOID
```

- 423 -

#define STOP 3

/* LookAhead histogram */

#define

HISTO

400

#define

HISTO_DELTA

20.0

#define

HISTO_BITS 9

source/Lex.l

```
%{
/*
      Lex driver for input files: .pal .vid .bat
*/
             "../include/xwave.h"
#include
#include
             "../include/Gram.h"
             ParseInput();
extern int
#undef
             unput
#undef
             input
#undef
             output
             feof
#undef
                          ungetc(c,global->parse_fp)
#define
             unput(c)
                                 ParseInput(global->parse_fp)
#define
             input()
#define
             output(c)
                          putchar(c)
#define
             feof()
                           (1) .
%}
             -?[0-9]+
number
             -?[0-9]+"."[0-9]+
fnumber
string \"([^"]|\\.)*\"
%start WAIT MAP VIDEO BATCH BATCH_TRANS BATCH_COMP
%n 2000
%p 4000
%e 2000
```

•

WO 94/23385

```
%%
                   c = '\0';
            char
                   while(c! = '/') {
                          while (c! = '*') c = input();
                          while (c = '*') c = input();
                   }
            }
      { BEGIN MAP; Dprintf("Lex: Reading palette file\n"); return(FILE_PAL); }
      { BEGIN VIDEO; Dprintf("Lex: Reading video file\n"); return(FILE_VID); }
\.vid
      { BEGIN BATCH; Dprintf("Lex: Reading batch file\n"); return(FILE_BAT); }
                   { (void)sscanf(yytext, "%d", &yylval.num); return(NUMBER); }
{mumber}
{string}
                   { yylval.ptr = (char *)yytext; return(STRING); }
                   { (void)sscanf(yytext, "%lf", &yylval.fnum); return(FNUMBER); }
{fnumber}
<MAP > Palette
                   { return(PALETTE); }
<MAP>\{
                          { return(LEFT_BRACE); }
<MAP>\}
                          { return(RIGHT_BRACE); }
<MAP>Range
                          { return(RANGE); }
<MAP>Line
                          { return(LINE); }
< VIDEO > Type
                          { return(TYPE); }
< VIDEO > MONO
                          { return(FORMAT_MONO); }
< VIDEO > RGB
                          { return(FORMAT_RGB); }
< VIDEO > YUV
                          { return(FORMAT_YUV); }
< VIDEO > Rate
                          { return(RATE); }
< VIDEO > Disk
                          { return(DISK); }
< VIDEO > Gamma { return(GAMMA); }
< VIDEO > Negative
                          { return(NEGATIVE); }
```

- 425 -

```
< VIDEO > Path
                        { return(PATH); }
< VIDEO > Files
                  { return(FILES); }
                        { return(TRANSFORM); }
< VIDEO > Transform
                  { return(TRANSFORM NONE); }
< VIDEO > None
< VIDEO > Wavelet { return(TRANSFORM_WAVE); }
< VIDEO > Start
                  { return(START); }
                        { return(END); }
< VIDEO > End
< VIDEO > Length { return(LEN); }
                        { return(DIM); }
< VIDEO > Dimensions
< VIDEO > Header { return(HEADER); }
< VIDEO > Offsets { return(OFFSETS); }
                        { return(SIZE); }
< VIDEO > Size
< VIDEO > Precision
                        { return(PRECISION); }
< VIDEO > Yes
                              { yylval.bool=True; return(BOOLEAN); }
< VIDEO > No
                              { yylval.bool=False; return(BOOLEAN); }
                              { return(LOAD); }
<BATCH > Load
                              { return(SAVE); }
<BATCH > Save
< BATCH > SaveAbekus
                        { return(SAVE_ABEKUS); }
<BATCH > Compare
                              { return(COMPARE); }
                              { return(DROP); }
<BATCH > Drop
                        { return(IMPORT_KLICS); }
< BATCH > ImportKLICS
                        { BEGIN BATCH TRANS; return(TRANSFORM); }
< BATCH > Transform
<BATCH > Compress
                              { BEGIN BATCH_COMP; return(COMPRESS); }
<BATCH > Xwave
                        { return(XWAVE); }
<BATCH > Shell
                        { return(SHELL); }
<BATCH > Copy
                              { return(COPY); }
<BATCH > Direct
                        { return(DIRECT_COPY); }
                              { return(DIFF); }
<BATCH > Diff
                              { return(LPF_WIPE); }
<BATCH > LPFzero
                              { return(LPF_ONLY); }
< BATCH > LPFonly
<BATCH > RGB-YUV
                              { return(RGB YUV); }
```

```
{ return(GAMMA); }
< BATCH > Gamma
                            { return(VIDEO_NAME); }
< BATCH COMP > VideoName
                             { return(STATS_NAME); }
< BATCH_COMP > Stats
                             { return(BIN_NAME); }
< BATCH_COMP > Binary
                                  { yylval.bool=True; return(BOOLEAN); }
< BATCH COMP > Yes
                                  { yyival.bool=False; return(BOOLEAN); }
<BATCH COMP>No
                             { return(STILL_MODE); }
< BATCH COMP > Still
                             { return(VIDEO_MODE); }
< BATCH COMP > Video
                             { return(AUTO_Q); }
<BATCH COMP > AutoQuant
                             { return(QUANT_CONST); }
< BATCH_COMP > QuantConst
< BATCH_COMP > ThreshConst
                            { return(THRESH_CONST); }
                             { return(BASE FACTOR); }
<BATCH COMP>BaseFactor
                             { return(DIAG_FACTOR); }
< BATCH COMP > DiagFactor
<BATCH_COMP>ChromeFactor { return(CHROME_FACTOR); }
< BATCH_COMP > Decision
                             { return(DECISION); }
< BATCH_COMP > Feedback
                             { return(FEEDBACK); }
                                   { return(DEC_MAX); }
<BATCH COMP>Maximum
                             { return(DEC_SIGABS); }
< BATCH_COMP > SigmaAbs
                             { return(DEC_SIGSQR); }
< BATCH_COMP > SigmaSqr
                             { return(FILTER); }
< BATCH_COMP > Filter
                             { return(FLT_NONE); }
< BATCH_COMP > None
                                   { return(FLT_EXP); }
< BATCH_COMP > Exp
                             { return(CMP_CONST); }
<BATCH COMP > CmpConst
                             { return(FPS); }
< BATCH_COMP > FrameRate
< BATCH_COMP > Bitrate
                             { return(BITRATE); }
                             { return(BUFFER); }
< BATCH_COMP > Buffer
                                   { return(LEFT_BRACE); }
<BATCH_COMP>\{
                                   { END; BEGIN BATCH;
<BATCH_COMP>\}
return(RIGHT BRACE); }
<BATCH_TRANS>VideoName { return(VIDEO_NAME); }
```

- 428 -

```
{ return(DIRECTION); }
< BATCH_TRANS > Direction
< BATCH_TRANS > Space { return(SPACE); }
                             { remrn(PRECISION); }
<BATCH_TRANS > Precision
<BATCH_TRANS > Yes
                             { yylval.bool=True; return(BOOLEAN); }
                                   { yylval.bool=False; return(BOOLEAN); }
<BATCH_TRANS > No
                                   { return(LEFT_BRACE); }
<BATCH_TRANS>\{
                             { END; BEGIN BATCH; return(RIGHT_BRACE); }
<BATCH_TRANS>\}
                 {;}
[. \t \n]
%%
yywrap() { return(1); }
```

source/Transform.h

```
typedef struct {
    Video src;
    char name[STRLEN], src_name[STRLEN];
    int space[2], precision;
    Boolean dirn;
} TransCtrlRec, *TransCtrl;
```

- 430 -

source/Video.h

```
typedef struct {
          char names[4][STRLEN];
} AbekusCtrlRec, *AbekusCtrl;
```

source/makefile

```
# Xwave Makefile
CFLAGS = -O -I../include
LIBS = -IXaw -IXmu -IXt -IXext -IX11 -Im -Il -L/usr/openwin/lib
.KEEP_STATE:
.SUFFIXES: .c .o
xwaveSRC = Select.c Convert.c xwave.c InitMain.c Pop2.c Video2.c Malloc.c
InitFrame.c \
             Frame.c Transform.c Convolve3.c Update.c Image.c Menu.c
PullRightMenu.c \
             NameButton.c SmeBSBpr.c Process.c Lex.c Gram.c Parse.c Color.c \
             Bits.c Storage.c Copy.c Message.c Palette.c ImportKlics.c Icon3.c Klics5.c
١
             KlicsSA.c KlicsTestSA.c ImportKlicsSA.c ImpKlicsTestSA.c
objDIR = .../\$(ARCH)
xwaveOBJ = (xwaveSRC: \%.c = (objDIR)/\%.o)
$(objDIR)/xwave: $(xwaveOBJ)
      gcc -o $@ $(xwaveOBJ) $(LIBS) $(CFLAGS)
$(xwaveOBJ): $$(@F:.o=.c) ../include/xwave.h
       gcc -c $(@F:.o=.c) $(CFLAGS) -o $@
Lex.c: Gram.c Lex.l
```

lex Lex.l mv lex.yy.c Lex.c

Gram.c: Gram.y

bison -dlt Gram.y

 $mv \ (@F:.c = .tab.h) ../include/Gram.h$

 $mv \ (@F:.c = .tab.c) Gram.c$

include/Bits.h

#endif

include/DTheader.h

```
typedef struct DTheader {
                            /* "DT-IMAGE" */
 char file_id[8];
                            /* 1 */
   char struct_id;
      char prod_id;
                                   /* 4 */
                                   /* 1 */
      char util id;
                                           /* 2 */
       char board_id;
       char create_time[9]; /* [0-1]year, [2]month, [3]dayofmonth, [4]dayofweek,
[5]hour, [6]min, [7]sec, [8]sec/100 */
                                   /* as create_time */
       char mod_time[9];
                                           /* 1 */
       char datum;
                                   /* 1024?? */
       char datasize[4];
                                   /* 1 */
       char file_struct;
                                           /* 1 */
       char datatype;
                                           /* 0 */
       char compress;
                                           /* 1 */
       char store;
                                           /* 4, 3 */
       char aspect[2];
                                           /* 8 */
       char bpp;
                                   /* 1 */
       char spatial;
                                           /* 512 */
       char width[2];
                                           /* 512 */
       char height[2];
                                    /* 512 */
       char full width[2];
       char full_height[2]; /* 512 */
       char unused1[45];
       char comment[160];
       char unused2[256];
} DTheader;
```

include/Icon.h

```
typedef
            enum {
      FW_label, FW_icon, FW_command, FW_text, FW_button, FW_icon_button,
FW_view, FW_toggle,
      FW_yn,
      FW_up, FW_down, FW_integer,
      FW_scroll, FW_float,
      FW_form,
} FormWidgetType;
            enum {
typedef
      SW_below, SW_over, SW_top, SW_menu,
} ShellWidgetType;
            struct {
typedef
      String name;
      String contents;
                   fromHoriz, fromVert;
      FormWidgetType
      String hook;
} FormItem;
```

include/Image.h

* \$XConsortium: Image.h,v 1.24 89/07/21 01:48:51 kit Exp \$
*/

Copyright 1987, 1988 by Digital Equipment Corporation, Maynard, Massachusetts, and the Massachusetts Institute of Technology, Cambridge, Massachusetts.

All Rights Reserved

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the names of Digital or MIT not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

DIGITAL DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING

ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL

DIGITAL BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR

ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS.

OURCESTIFF OURSE /DIES AC

WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION.

ARISING OUT	OF OR IN C	CONNECTION	WITH	THE USE O	R PERFORMANCE OF
THIS					
SOFTWARE.					
****	******	******	*****	****	*****
#ifndef _XawIn	-				
#define _XawIn	nage_h				·
/ 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4	******	****	****	****	*****

-					
* Image Widge	et				
*					
****	****	*****			,
#include <x11< td=""><td>/Yaw/Simale</td><td>h></td><td></td><td></td><td></td></x11<>	/Yaw/Si mal e	h>			
#include <x11< td=""><td>_</td><td></td><td></td><td>•</td><td></td></x11<>	_			•	
#include < XII	I/Amu/Conve	11613.11			
/* Resources:					
Name	Cla	ass	RepTy	pe	Default Value
****		******			•
border	Во	rderColor	Pixel	XtDef	faultForeground
borderWidth	BorderWi	dth Dimer	nsion	1	
cursor	Cu	rsor	Cursor	•	None
destroyCallbac	k Callbac	k	XtCall	backList	NULL
insensitiveBor	der Insensiti	ve Pixma	ıp	Gray	
mappedWhenl	Managed Ma	appedWhenMa	naged	Boolean	True
sensitive	Sensitive	Boole	an	True	
bitmap	Bitmap	Pixmap	NULL	,	
callback	Caliback	XtCallbackLi	ist	NULL	
	Docition	Pociti	OB	0	

- 438 -

Position 0 Position y */ #define XtNbitmap "bitmap" #define XtCBitmap "Bitmap" /* Class record constants */ extern WidgetClass imageWidgetClass; typedef struct ImageClassRec *ImageWidgetClass; typedef struct _ImageRec *ImageWidget; #endif /* _XawImage_h */ /* DON'T ADD STUFF AFTER THIS #endif */

include/ImageHeader.h

```
/* Author: Philip R. Thompson
   Address: phils@athena.mit.edu, 9-526
   Note: size of header should be 1024 (1K) bytes.
   $Header: ImageHeader.h,v 1.2 89/02/13 09:01:36 phils Locked $
   $Date: 89/02/13 09:01:36 $
   $Source: /mit/phils/utils/RCS/ImageHeader.h,v $
*/
#define IMAGE_VERSION
typedef struct ImageHeader {
   char file_version[8]; /* header version */
                         /* Size of file header in bytes */
   char header_size[8];
                          /* Width of the raster image */
   char image_width[8];
   char image_height[8]; /* Height of the raster imgage */
                          /* Actual number of entries in c_map */
   char num colors[8];
                           /* 0 or 1 = pixmap, 3 = RG&B buffers */
   char num channels[8];
   char num pictures[8];
                          /* Number of pictures in file */
   char alpha_channel[4]; /* Alpha channel flag */
                         /* Runlength encoded flag */
   char runlength[4];
                         /* Name of who made it */
   char author[48];
                        /* Date and time image was made */
   char date[32];
                          /* Program that created this file */
   char program[16];
                           /* other viewing info. for this image */
   char comment[96];
   unsigned char c_map[256][3]; /* RGB values of the pixmap indices */
} ImageHeader;
```

/* Note:

* - All data is in char's in order to maintain easily portability

- 440 -

- * across machines and some human readibility.
- * Images may be stored as pixmaps or in seperate channels, such as
- * red, green, blue data.
- * An optional alpha channel is seperate and is found after every
- num channels of data.
- * Pixmaps, red, green, blue, alpha and other channel data are stored
- * sequentially after the header.
- * If num_channels = 1 or 0, a pixmap is assumed and up to num_colors
- * of colormap in the header are used.

*/

/*** end ImageHeader.h ***/

/*

include/ImageP.h

* \$XConsortium: ImageP.h,v 1.24 89/06/08 18:05:01 swick Exp \$

/****************************

Copyright 1987, 1988 by Digital Equipment Corporation, Maynard, Massachusetts, and the Massachusetts Institute of Technology, Cambridge, Massachusetts.

All Rights Reserved

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the names of Digital or MIT not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

DIGITAL DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING

ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL

DIGITAL BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR

ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,

WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER

- 442 -

TORTIOUS ACTION,
ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF
THIS .
SOFTWARE.

/*
* ImageP.h - Private definitions for Image widget
*/
William V.
#ifndef _XawImageP_h
#define _XawImageP_h
/*****************
•
* Image Widget Private Data
•

#include "/include/Image.h"
#include <x11 simplep.h="" xaw=""></x11>
/* New fields for the Image widget class record */
typedef struct {int foo;} ImageClassPart;
/* Full class record declaration */
typedef struct _ImageClassRec {
CoreClassPart core_class;
SimpleClassPart simple class;

```
ImageClassPart
                    image_class;
} ImageClassRec;
extern ImageClassRec imageClassRec;
/* New fields for the Image widget record */
typedef struct {
   /* resources */
      Pixmap
                    pixmap;
      XtCallbackList
                           callbacks;
   /* private state */
      Dimension
                    map_width, map_height;
} ImagePart;
 * Full instance record declaration
typedef struct _ImageRec {
   CorePart core;
   SimplePart
                    simple;
   ImagePart image;
} ImageRec;
#endif /* _XawImageP_h */
```

- 444 -

include/Message.h

```
typedef struct {
    Widget shell, widget; /* shell and text widgets (NULL if not created */
    XawTextBlock info; /* Display text */
    int size, rows, cols; /* Size of buffer (info.ptr) & dimensions of display */
    XawTextEditType edit; /* edit type */
    Boolean own_text; /* text is owned by message? */
} MessageRec, *Message;
```

include/Palette.h

```
#define PalettePath "."

#define PaletteExt ".pal"

typedef struct _MapRec {
    int start, finish, m, c;
    struct _MapRec *next;

} MapRec, *Map;

typedef struct _PaletteRec {
    char name{STRLEN};
    Map mappings;
    struct _PaletteRec *next;

} PaletteRec, *Palette;
```

include/PullRightMenu.h

/*

* \$XConsortium: PullRightMenu.h,v 1.17 89/12/11 15:01:55 kit Exp \$

.

* Copyright 1989 Massachusetts Institute of Technology

*

- * Permission to use, copy, modify, distribute, and sell this software and its
- * documentation for any purpose is hereby granted without fee, provided that
- * the above copyright notice appear in all copies and that both that
- * copyright notice and this permission notice appear in supporting
- * documentation, and that the name of M.I.T. not be used in advertising or
- * publicity pertaining to distribution of the software without specific,
- * written prior permission. M.I.T. makes no representations about the
- * suitability of this software for any purpose. It is provided "as is"
- * without express or implied warranty.

*

- * M.I.T. DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL
- * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL M.I.T.
- * BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
- * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION
- * OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN
- * CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

. */

/* * PullRightMenu.h - Public Header file for PullRightMenu widget. * This is the public header file for the Athena PullRightMenu widget. * It is intended to provide one pane pulldown and popup menus within * the framework of the X Toolkit. As the name implies it is a first and * by no means complete implementation of menu code. It does not attempt to * fill the needs of all applications, but does allow a resource oriented * interface to menus. */ #ifndef PullRightMenu_h #define PullRightMenu_h #include <X11/Shell.h> #include <X11/Xmu/Converters.h> * PullRightMenu widget /* PullRightMenu Resources: Default Value Class RepType Name **XtDefaultBackground Pixel** Background background None BackgroundPixmap Pixmap backgroundPixmap XtDefaultForeground **Pixel** BorderColor borderColor None Pixmap BorderPixmap borderPixmap

*/

- 448 -

borderWidth	BorderWidth	Dimension	1	•
bottomMargin	VerticalMarg	ins Dimensio	on Ve	rticalSpace
columnWidth	ColumnWidth	Dimens	ion W	idth of widest text
cursor	Cursor	Cursor	None	
destroyCallback	Callback	Pointe	er	NULL
height	Height	Dime	nsion 0	
label	Label	String	NULL (No	label)
labelClass	LabelClass	Pointer	smeBSB	Object Class
mappedWhenMa	anaged MappedV	VhenManaged	Boolean	True
rowHeight	RowHeight	Dimensio	n Heig	ght of Font
sensitive	Sensitive	Boolean	Tr	ue
topMargin	VerticalMargin	s Dimension	n Vert	icalSpace
width	Width	Dimension	0	
button	Widget	Widget NUL	L	
x	Position	Position	0	
y	Position	Position	0	

typedef struct _PullRightMenuClassRec* PullRightMenuWidgetClass;
typedef struct _PullRightMenuRec* PullRightMenuWidget;

extern WidgetClass pullRightMemuWidgetClass;

#define XtNcursor "cursor"

#define XtNbottomMargin "bottomMargin"

#define XtNcolumnWidth "columnWidth"

#define XtNlabelClass "labelClass"

#define XtNmenuOnScreen "mcnuOnScreen"

#define XtNpopupOnEntry "popupOnEntry"

#define XtNrowHeight "rowHeight"

#define XtNtopMargin "topMargin"

```
"button"
#define XtNbutton
#define XtCColumnWidth "ColumnWidth"
#define XtCLabelClass "LabelClass"
#define XtCMenuOnScreen "MenuOnScreen"
#define XtCPopupOnEntry "PopupOnEntry"
#define XtCRowHeight "RowHeight"
#define XtCVerticalMargins "VerticalMargins"
                          "Widget"
            XtCWidget
#define
 * Public Functions.
/*
       Function Name: XawPullRightMenuAddGlobalActions
       Description: adds the global actions to the simple menu widget.
       Arguments: app_con - the appcontext.
       Returns: none.
 */
void
XawpullRightMenuAddGlobalActions(/* app_con */);
XtAppContext app_con;
*/ .
#endif /* _PullRightMenu_h */
```

- 450 -

include/SmeBSBpr.h

/*

* \$XConsortium: SmeBSB.h,v 1.5-89/12/11 15:20:14 kit Exp \$

#

* Copyright 1989 Massachusetts Institute of Technology

- * Permission to use, copy, modify, distribute, and sell this software and its
- * documentation for any purpose is hereby granted without fee, provided that
- * the above copyright notice appear in all copies and that both that
- * copyright notice and this permission notice appear in supporting
- * documentation, and that the name of M.I.T. not be used in advertising or
- * publicity pertaining to distribution of the software without specific,
- * written prior permission. M.I.T. makes no representations about the
- * suitability of this software for any purpose. It is provided "as is"
- * without express or implied warranty.

*

- * M.I.T. DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL
- * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL M.I.T.
- * BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
- * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION
- * OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN
- * CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

*/

```
* SmeBSBpr.h - Public Header file for SmeBSB object.
 * This is the public header file for the Athena BSB Sme object.
 * It is intended to be used with the simple menu widget. This object
 * provides bitmap - string - bitmap style entries.
 */
#ifndef _SmeBSBpr_h
#define _SmeBSBpr_h
#include <X11/Xmu/Converters.h>
#include <X11/Xaw/Sme.h>
  SmeBSBpr object
/* BSB pull-right Menu Entry Resources:
 Name
                        Class
                                        RepType
                                                             Default Value
                 Callback
callback
                                 Callback
                                                NULL
destroyCallback
                   Caliback
                                        Pointer
                                                             NULL
font
                Font
                                XFontStruct * XtDefaultFont
foreground
                  Foreground
                                    Pixel
                                                 XtDefaultForeground
height
                       Height
                                        Dimension
label
              Label
```

String

Name of entry

leftBitmap	LeftBitmap	Pixmap	None
leftMargin	HorizontalMar	gins Dimensio	on 4
rightBitmap	RightBitmap '	Pixmap	None
rightMargin	HorizontalMa	gins Dimensio	on 4
sensitive	Sensitive	Boolean	True
vertSpace	VenSpace	int	25
width	Width	Dimension	0
x	Position	Position	0n
у	Position	Position	0
menuName	MenuName String	g "menu"	

. +/

extern WidgetClass smeBSBprObjectClass;

#define XtNleftBitmap "leftBitmap"

#define XtNleftMargin "leftMargin"

#define XtNrightBitmap "rightBitmap"

#define XtNrightMargin "rightMargin"

#define XtNvertSpace "vertSpace"

#define XtNmenuName "menuName"

#define XtCLeftBitmap "LeftBitmap"
#define XtCHorizontalMargins "HorizontalMargins"

#define XtCRightBitmap "RightBitmap"

#define XtCVertSpace "VertSpace"

#define XtCMenuName "MenuName"

#endif /* _SmeBSBpr_h */

include/SmeBSBprP.h

/*

- * \$XConsortium: SmeBSBP.h,v 1.6 89/12/11 15:20:15 kit Exp \$
- * Copyright 1989 Massachusetts Institute of Technology
- * Permission to use, copy, modify, distribute, and sell this software and its
- * documentation for any purpose is hereby granted without fee, provided that
- * the above copyright notice appear in all copies and that both that
- * copyright notice and this permission notice appear in supporting
- * documentation, and that the name of M.I.T. not be used in advertising or
- * publicity pertaining to distribution of the software without specific,
- * written prior permission. M.I.T. makes no representations about the
- * suitability of this software for any purpose. It is provided "as is"
- * without express or implied warranty.

*

- * M.I.T. DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL
- * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL M.I.T.
- * BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
- * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION
- * OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN
- * CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
- * Author: Chris D. Peterson, MIT X Consortium

```
* SmeP.h - Private definitions for Sme object
+/
#ifndef XawSmeBSBP_h
#define XawSmeBSBP_h
 * Sme Object Private Data
#include <X11/Xaw/SmeP.h>
#include "../include/SmeBSBpr.h"
 * New fields for the Sme Object class record.
 typedef struct SmeBSBprClassPart {
  XtPointer extension;
 } SmeBSBprClassPart;
 /* Full class record declaration */
 typedef struct _SmeBSBprClassRec {
                         rect_class;
    RectObjClassPart
```

```
SmeClassPart
                    sme class;
   SmeBSBprClassPart sme_bsb_class;
} SmeBSBprClassRec;
extern SmeBSBprClassRec smeBSBprClassRec;
/* New fields for the Sme Object record */
typedef struct {
   /* resources */
                            /* The entry label. */
   String label;
                            /* extra vert space to leave, as a percentage
   int vert space;
                               of the font height of the label. */
   Pixmap left_bitmap, right_bitmap; /* bitmaps to show. */
   Dimension left_margin, right_margin; /* left and right margins. */
                            /* foreground color. */
   Pixel foreground;
                                   /* The font to show label in. */
   XFontStruct * font;
   XtJustify justify;
                            /* Justification for the label. */
       String menu_name; /* Popup menu name */
/* private resources. */
   Boolean set_values_area_cleared; /* Remember if we need to unhighlight. */
                                    /* noral color gc. */
    GC norm_gc;
                                    /* reverse color gc. */
    GC rev_gc;
                                    /* Normal color (grayed out) gc. */
    GC norm_gray_gc;
                            /* gc for flipping colors. */
    GC invert_gc;
    Dimension left_bitmap_width; /* size of each bitmap. */
    Dimension left_bitmap_height;
    Dimension right_bitmap_width;
    Dimension right bitmap_height;
```

SmeBSBprran;
/ ************************************
• .
* Full instance record declaration
•

typedef struct _SmeBSBprRec {
ObjectPart object;
RectObjPart rectangle;
SmePart sme;
SmeBSBprPart sme_bsb;
} SmeBSBprRec;
·
/+++++++++++++++++++++++++++++++++++++
*
* Private declarations.

#endif /* _XawSmeBSBPpr_h */

include/xwave.h

< X11/Xlib.h> #include <X11/Xutil.h> #include <X11/Xatom.h> #include <X11/Xaw/Cardinals.h> #include <X11/StringDefs.h> #include <X11/Xmu/Xmu.h> #include <X11/Xaw/Command.h> #include <X11/Xaw/List.h> #include <X11/Xaw/Box.h> #include <X11/Xaw/Form.h> #include <X11/Xaw/Scrollbar.h> #include <X11/Xaw/Viewport.h> #include <X11/Xaw/AsciiText.h> #include #include <X11/Xaw/Dialog.h> <X11/Xaw/MenuButton.h> #include <X11/Xaw/SimpleMenu.h> #include <X11/Xaw/SmeBSB.h> #include <X11/Xaw/Toggle.h> #include "SmcBSBpr.h" #include "PullRightMenu.h" #include <X11/Shell.h> #include <X11/cursorfont.h> #include **STRLEN** 100 #define NAME_LEN 20 #define #include "Image.h" "Message.h" #include <dirent.h> #include

< math.h>

#include

```
< stdio.b>
#include
            "Palette.h"
#include
            "Icon.h"
#include
                        "graphs"
#define
            PLOT_DIR
            PLOT_EXT ".plot"
#define
            ELLA_IN_DIR
#define
#define
            ELLA IN EXT
            ELLA_OUT_DIR
#define
             ELLA OUT_EXT
#define
             VID_DIR
                          "videos"
#define
#define
             VID EXT
                          ".vid"
             IMAGE_DIR "images"
#define
             BATCH DIR "batch"
#define
#define BATCH_EXT
                          ".bat"
             KLICS_DIR "import"
#define
             KLICS EXT ".klics"
#define
             KLICS_SA_DIR
                                "import"
#define
             KLICS_SA_EXT
                                ".klicsSA"
#define
typedef enum {
       TRANS None, TRANS_Wave,
} TransType;
             emm {
typedef
       MONO, RGB, YUV,
 } VideoFormat;
 extern String ChannelName[3][4];
             negif(bool, value)
                                ((bool)?-(value):(value))
 #define
```

```
rypedef
             struct {
       String name;
       Pixmap
                    pixmap;
      unsigned int height, width;
} lconRec, *lcon;
                   (*Proc)();
typedef
             void
typedef
             String *(*ListProc)();
typedef
             Boolean
                           (*BoolProc)();
             struct {
typedef
       String name;
       WidgetClass widgetClass;
       String label;
       String hook; /* menuName for smeBSBprObjectClass */
} MenuItem;
             struct {
typedef
      String name, button;
      ListProc
                    list proc;
      String action_name;
      Proc action_proc;
       caddr_t
                    action_closure;
} SelectItem, *Selection;
typedef
             struct {
      TransType
                    type;
             space[2];
       int
       Boolean
                    dirn;
} WaveletTrans;
typedef
             union {
```

```
TransType
                     type;
       Wavelet Trans
                            wavelet;
} VideoTrans;
typedef
             struct VideoRec
                                                 /* Name of this video name.vid */
       char
             name(STRLEN);
       char
             path[STRLEN];
                                                        /* Path to frame file(s) */
                                          /* Name of frames files001 if not name */
       char
             files[STRLEN];
                                          /* Type of video (MONO, RGB, YUV) */
       VideoFormat type;
                     disk; /* Frames reside on disk rather than in memory */
       Boolean
                                                        /* Gamma corrected flag */
       Boolean
                     gamma;
       Boolean
                                                 /* Load negative values in data */
                     negative;
                                                        /* Frames per second */
       int
              rate;
       int
             start:
                                                 /* Starting frame number */
             size[3]; /* Dimensions of video after extraction x, y and z */
       int
             UVsample[2];
                                          /* Chrominance sub-sampling x and y */
       int
       int
             offset:
                                          /* Header length */
                                          /* Dimensions of video as stored */
      int
             cois, rows:
       int
             x_offset, y_offset; /* Offset of extracted video in stored */
       VideoTrans trans;
                                                 /* Transform technique used */
      int
             precision;
                                          /* Storage precision above 8 bits */
                                                 /* Image data channels */
       short **data[3];
      struct VideoRec
                                                 /* Next video in list */
                            *next:
} VideoRec, *Video;
typedef
             struct {
      Video video:
             name[STRLEN];
      char
} VideoCtrlRec, *VideoCtrl;
             struct PointRec
typedef
      int
             location[2];
```

```
int
            usage;
      struct PointRec
                          *next;
} PointRec, *Point;
            struct FrameRec {
typedef
      Widget
                   shell, image widget, point_merge_widget;
      Video video;
            zoom, frame, channel, palette;
                   point switch, point merge;
      Boolean
      Point point;
      Message
                   msg;
      struct FrameRec *next;
} FrameRec, *Frame;
            NO_CMAPS 6
#define
             struct BatchRec
typedef
      Proc proc;
      caddr t
                   closure, call_data;
      struct BatchRec
                          *next;
} BatchRec, *Batch;
typedef
             struct {
      char home[STRLEN];
      XtAppContext
                          app_con;
      Widget
                   toplevel;
             no_icons;
      int
      Icon icons;
       Video videos;
       Frame frames;
      Point points;
       Palette
                   palettes;
```

```
int
             no_pals;
      String parse_file;
      String parse_token;
      FILE *parse_fp;
      XVisualInfo *visinfo;
             levels, rgb_levels, yuv_levels[3];
       int
                    cmaps[NO_CMAPS];
       Colormap
      String batch;
      Batch batch_list;
       Boolean
                    debug;
             dither[16][16];
       int
} GlobalRec, *Global;
             struct {
typedef
       Widget
                    widgets[3];
       int
             max, min, *value;
       String format;
} NumInputRec, *NumInput;
typedef
             struct {
       Widget
                    widgets[2];
                    max, min, *value;
      double
       String format;
} FloatInputRec, *FloatInput;
extern Global
                    global;
/* InitFrame.c */
extern Video FindVideo();
/* Pop2.c */
```

```
extern void NA();
                    FindWidget();
extern Widget
extern void
             Destroy();
            Free();
extern void
/* Storage.c */
             NewFrame();
extern void
extern void
             GetFrame();
             SaveFrame();
extern void
extern void
            FreeFrame();
             SaveHeader();
extern void
extern Video CopyHeader();
/* Message.c */
extern void
             TextSize();
                    NewMessage();
extern Message
            MessageWindow();
extern void
extern void CloseMessage();
            Mprintf();
extern void
extern void
            Dprintf();
            Eprintf();
extern void
            Mflush();
extern void
/* Icon3.c */
extern void
             FillForm();
extern void
             FillMenu();
                    ShellWidget();
extern Widget
extern Widget
                    FormatWidget();
extern void SimpleMenu();
```

extern int TextWidth();
extern Icon FindIcon();
extern void NumIncDec();
extern void FloatIncDec();
extern void ChangeYN();

extern XFontStruct *FindFont();

DATA COMPRESSION AND DECOMPRESSION
GREGORY KNOWLES AND ADRIAN S. LEWIS
M-2357 US
APPENDIX B-1

```
MAC ADDR_COUNTER_COL = (bool:ck,t_reset:reset,STRING[xsize]bit:block_cnt_length)
```

(t_col,bool): BEGIN

->base_counter_col. MAKE BASE_COUNTER_COL:base_counter_col.
JOIN (ck,reset,block_cnt_length) ->base_co

OUTPUT (base_counter_col[1], CASE base_counter_col[2]
OF count carry:1 count_carry:t

ELSE 1 ESAC)

END.

MAC ADDR_COUNTER_ROW = (bool:ck,t_reset:reset,STRING[ysize]bit:block_cnt_length,bool:col_carry)

(t_row,bool):

BEGIN

BASE_COUNTER_ROW:base_counter_row. MAKE

#type conversion# JOIN (ck,reset,col_carry,block_cnt_length,CASE col_carry
OF t:count_carry

ELSE count_rst ESAC) ->base_counter_row.

OUTPUT (base_counter_row[1], CASE base_counter_row[2] count_camy.t

ELSE 1

ESAC)

END.

#the string base address calculators#

MAC NOMULT_MAC_READ = (bool:ck,t_reset:reset,bool:col_end,t_mux4:mux_control,STRING[17]bit:incr, STRING[17]bit:incr, STRING[17]bit:base_v)

STRING[19]bit:

BEGIN

MAKE ADD_US_ACTEL{19,17}:add, MUX_2{STRING{17}bit}:mux.

next_addr = MUX_4{STRING(19|bit){add(2..20],ZERO(19|b*0*,base_u,base_v,mux_control), dff = DFF_NO_LOAD(STRING(19|bit){ck,reset,next_addr,b*000000000000000000000}). LET

->add, (dff,mux,b'1) N_O

(incr,oct_add_factor,CASE col_end

OF tinght

ELSE left ESAC)

->**≡**EX.

青 OUTPUT

END.

MAC S_SPA =(STRING[19]bit:in)

(flag,t_sparc_addr):BIOP TRANSFORM_US. MAC SPA_S =(t_sparc_addr:in)

(flag, STRING[19]bit):BIOP TRANSFORM_US.

MAC SPARC_ADDR= (bool:ck,t_reset:reset:reset.bool:col_end,t_mux4:mux_control,[2]t_sparc_addr:oct_add_factor,

STRING[19]bit.base_u base_v)

BEGIN

LET out=NOMULT MAC READ(ck,reset,col_end,mux_control,(SPA_S oct_add_factor[1])[2][3..19]

t_sparc_addr:

(SPA_S oct_add_factor[2])[2][3..19],base_u,base_v).

#

#the read and write address generator,input the initial image & block sizes for oct/0 at that channel# FN ADDR_GEN_NOSCRATCH= (boot:ck,t_reset:reset,t_direction:direction,t_channel:channel

STRING[9]bit x p_1,STRING[11]bit x3 p_1,STRING[12]bit:x7 p_1,

STRING [ysize]bit:octave_row_length,STRING [xsize]bit:octave_col_length,t_reset:octave_reset,

Loctave:octave,booly_done,bool:uv_done, t_bad:octave_finished, STRING [19]bit:base_u base_v)

((1_input_mux,t_sparcport,t_dwtport#dwt#),t_load#IDWT data valid#,t_load#read_valid# !_count_control#row read col read#,(!_col,!_count_control)#addr_col_read#): #the current octave and when the block finishes the 3 octave transform#

ADDR_COUNTER_ROW:addr_row_write,#

ADDR_COUNTER_COL.addr_col_write,#

MAKE

ROW_COUNT_CARRY:addr_row_read, COL_COUNT: addr_col_read, SPARC_ADDR:write_addr_read_addr,

MEM CONTROL NOSCRATCH: mam_control,

#write
ne_bool.
ol read_done_bool
JKFF:zero_hh_bool
JKFF:zer

= CASE octave oct/0:mo, P

mem_se

LET

oct/1:dos,

oct/2:tres,

oct/3:quatro

ESAC,

= MUX_4(t_sparc_addr)(

sparc_add_1

(addr/1),

(addr/2),

(addr/4),

mem_sel), (addr/8),

- MUX_4(STRING[12|5it](spare add 2 y

(b'000" CONC x_p_1[1...7] CONC b'10") (b'0" CONC x3_p_1[1...8] CONC b'100") (x7_p_1[1..8] CONC b'1000"), (5.00000000001,)

mem_sel),

= MUX_4(STRING[12]bit)((P.000000000011) spare_add_2_uv

(b'0000" CONC x_p_1[1..6] CONC b'10")

(b'00 CONC x3 p 1(1...7) CONC b'100"), (b'0" CONC x7 p 1(1...7) CONC b'1000"),

mem_sel),

sparc_add_2

= MUX_2{STRING[12]bit}{ spare_add_2_y, spare_add_2_uv, CASE channel

y:left right ELSE ESAC),

sparc_oct_add_factor = (sparc_add_1,(S_SPA(b*0000000* CONC sparc_add_2))[2]),

#signals when write must start delayed 1 to for use in zero_hh#

addr_col_read_flag =CASE addr_col_read[2]#decode to bool#

OF count_carry.t ELSE (ESAC,

write_latency = CASE (addr_row_read[1], addr_col_read[1])
OF (row/2,col/(conv2d_latency-1)):t
ELSE t

ESAC,

#read input data done# read_done = CASE (addr_row_read[2], addr_col_read_flag)
OF (count_carry,t):t
ELSE f
ESAC,

zero_hh = CAST(t_load)(NOT zero_hh_boot),

read_valid= CAST(t_load)(NOT read_done_bool),

start_write_col= DFF_NO_LOAD(t_load)(dk,reset,zero_hh,read),

#1 tu after zero_hh#

#base y# Pase_u# #pase n# (y_done,uv_done,octave_finished,channel) (t,f,write,y)[(f,f,write,u):tres, #base_u# #base v# fbase y# (f,t,write,u)[(f,f,write,v):qualro, utres, CASE channel y:dos, (f,bool,write,y):dos 2 CASE zero_th read: CASE () ELSE ESAC, write: uno, read_mux = write_mux =

ESAC

tbase v#

v:quatro

ESAC

(ck,octave_reset,write_latency,t) ->zero_hh_boot,

#the row&col counts for the read address#

->addr_row_read,

#note that all the counters have to be reset at the end of an octave, le on octave_finished#

(ck,octave_reset,octave_col_length) ->addr_col_read, (ck,octave_reset,octave_row_length,addr_col_read(2))

->read_done_bool,

(ck.octave_reset.read_done,t)

(ck,reset,PDF1[bool,conv2d_latency-1](ck,reset,addr_col_read_flag,f),write_mux,sparc_oct_add_factor,base_u,base_v) #w&r addresses for sparc mem# ->write_addr,

NOS

(ck,reset,addr_col_read_flag,read_mux,sparc_oct_add_factor,base_u,base_v) ->read_addr,

(ck,reset,direction,channel,octave,write_addr,read_addr,zero_hh)->mem_control

OUTPUT(mem_control,zero_hh, read_valid,addr_row_read[2],addr_col_read)

END.

#the basic 2d convolver for transform, rows first then cols.#

FN CONV_2D = (bool:ck,t_reset.reset, t_input:in, t_direction:direction, [4]t_scratch:pdel,

1_reset.conv_reset,1_count_control.row_flag,(1_cot,1_count_control).addr_col_read)

(Linput, Limemport, Lount control, Lount control, Lount control):

#forward direction outputs in row form #

нн нд нн нд

HG GG HG GG..... HH HG HH HG

HG GG HG GG.....

#the inverse convolver returns the raster scan format output data#

#the convolver automatically returns a 3 octave transform#

BEGIN

FN CH_PORT = ([[4]t_scratch,t_col),t_col)

t_memport:REFORM

MAKE CONV_ROW:conv_row, CONV_COL:conv_col.

LET

```
#pipeline delays in col_conv#
                                                                                                                                                                  #pipeline delays in row_conv#
                                                                                                                                    OF torward:PDF1(I_reset,3)(ck.no_rst,conv_reset,rst),
                                                      inverse: PDF1 [t_reset, 1] (ck, no_rst, conv_reset, rst)
                                                                                                                                                                inverse: conv_reset
                                                                                                          CASE direction
                             OF forward:conv_reset,
row_reset = CASE direction
                                                                                 ESAC,
                                                                                                              col reset =
```

col_flag = DFM(t_count_control)(ck,addr_col_read[2],PDF1(t_count_control,1)(dk,reset,addr_col_read[2], count 0), CAST(bool)direction)

row_control = DFM(t_count_control)(ck,PDF1(t_count_control,3)(ck,reset,row_flag,count_0), row_flag, CAST(bool)direction),

direction_sel =CASE direction #mux control for the in/out data mux's# OF forward:left, inverse:right ESAC, col_count = MUX_2{(I_col,t_count_control)}{
PDF1{(t_col,t_count_control),3}{ck,reset,addr_col_read,(col/0,count_rst)}
addr_col_read,
direction_sel),

#pipeline delays for the convolver values and input value# del_conv_col=DFF_NO_LOAD(t_input)(ck,reset,conv_co[[1],input/0),

del_conv_row=DFF_NO_LOAD(t_input)(ck,reset,conv_row,input/0),

del_in = DFF_NO_LOAD(t_input)(ck,reset,in,input/0)

(ck,row_reset,direction,MUX_2(t_input)(del_in,del_conv_col,direction_sel), col_flag) ->conv_row,

(ck,col_reset,direction,MUX_2(t_input)(del_conv_row,del_in,direction_sel), pdel,row_control,col_count) ->conv_col.

OUTPUT (MUX_2(t_input){del_conv_col,del_conv_row, direction_sel), CH_PORT(conv_col[2],col_count[1]),row_control,col_count[2],col_flag)

1d col convolver, with control

FN CONV_COL = (bool:ck,t_reset:reset, t_direction:direction, t_input:in,

[4]_scratch:pdel,t_count_control:row_flag,

(1_col.t_count_control):col_count)

(t_input,([4]t_scratch,t_col)):

out is (G,H), and line delay out port. The row counter is started 1 cycle later to allow for# #input is data in and, pdel, out from line-delay memories#

#pipeline delay between MULTIPLIER and this unit #

a %2 line by line resetable counter for the state machines, out->one on ret# #carry active on last element of row#

MAC COUNT_2 = (bool:ck,t_reset:reset,t_count_control:carry)

t_count_2:

BEGIN

MAKE DFF_NO_LOAD[i_count_2]:countdel.countout= CASE (countdel,carry) 回

(one,count_carry):two, (two,count_carry):one

ELSE countdel

ESAC.

JOIN (ck.reset.countout.one) ->countdel

OUTPUT countdel

MAKE MULT ADD:mult add #the code for the convolver#

[4]DF1(f_seratch):pdel_out, COUNT_2:count. [4]DF1(t_scratch):pdel_in,

now the state machines to control the convolver# #First the and gates#

E

#starts row counter 1 cycle after frame start# #we want the row counter to be 1 cycle behind the col counter for the delay for the reset_row=DF1(t_reset)(ck,reset) #pipelined line delay memory#

col_carry =DFF_NO_LOAD(I_count_control)(ck,reset,col_count[2],count_ret),

#these need to be synchronised to keep the row counter aligned with the data stream# #also the delay on col_count deglitches the col carryout#

row_control=row_flag,

#signal for row=0,1,2,3, last row, etc#

andsel=(CASE direction OF forward: CASE count

one:pass, two:zero

one:zero, ESAC, inverse: CASE count OF

two:pass

ESAC

ESAC,

CASE row_control OF count_0'zero

ELSE pass ESAC,

CASE row co...
OF count_0zero
ELSE pass
ESAC,
inverse: pass **CASE** direction P

#now the add/sub control for the convolver addens# addsel= CASE count

OF one:(add,add,add,sub), two:(add,sub,add,add) ESAC,

centermuxsel= #now the mux control#

CASE direction OF forward:

forward: CASE count

OF one:(lett,right), two:(right,left ESAC,

one:(right,left) two:(left,right) inverse:CASE count Р

ESAC

ESAC,

#the perfect reconstruction output# #the addmuxsel signal#

CASE direction

muxandsel =

forward:(andsel[2],pass,andsel[2]), inverse:(pass,andsel[2], CASE row_control

count 1 zero

ELSE pass ESAC)

ESAC, **CASE** direction

forward:(uno, muxsel=

CASE row_control OF count_0:dos,

count_carry:tres ELSE uno

ESAC,

CASE row_control OF count_0.tres,

count_carry:quatro ELSE dos

ESAC).

inverse: (CASE row_control OF count 0:dos,

count_carry:dos, count_1:quatro,

count Im1 tres

ELSE dos ESAC,

CASE row_control OF count_0.tres,

count_carry:dos EI.SE uno

ESAC,

Our

ESAC.

#ACTEL#

wr_addr

rd_addr=col_count[1].

=DF1{t_col}(ck,DF1{t_col}(ck,col_count[1])), #need 2 delays between wr and rd addr#

#address for line delay memory#

#join the control signals to the mult_add block# JOIN (ck,reset_row,col_carry)->count. (ck,reset_row,col_carry) ->count,

->mult_add. (ck, reset, in, and set, centermux set, mux set, mux and set, addset, direction, putel_out)

```
#read delay to match MULT delay#
                         #delay to catch the write address#
                         (ck,mult_add[k]) ->pdel_in[k], (ck,pdel[k]) ->pdel_out[k].
FOR INT k=1..4 JOIN
```

LET gh_select = CASE (direction,DF1(t_count_2)(ck,count) | (irverse,one)|(forward,two):right, (inverse, two)|(forward, one):left #ACTEL HACK# ESAC, gh_out = MUX_2(t_scratch)(pdet_in[4],DF1(t_scratch)(ck,pdet_out[1]),gh_select), shift const= CASE direction

CASE DF1[L_count_control](ck,row_control) OF inverse:

(count_1 | count_2):shift3

ELSE shift4

ESAC,

shiff5 forward: OUTPUT (ROUND_BITS(gh_out,shift_const), (pdel_in,wr_addr#rd_addr#))

#the 1d convolver, with control and coeff extend#

FN CONV_ROW =(bool:ck,t_resetzeset,t_drection:drection,t_input.in, t_count_control:col_flag)

out is (G,H). The row counter is started 1 cycle later to allow for# #the strings give the col & row lengths for this octave# #pipeline delay between MULTIPLIER and this unit

a %2 line by line resetable counter for the state machines, out->one on rst#

MAC COUNT_2 = (bool:ck,t_reset:reset)

BEGIN

MAKE DFF_NO_LOAD(t_count_2):countdel.countout= CASE (countdel)

t_count_2:

LET

(one):two,

two):one

ESAC.

JOIN (ck,reset,countout,one) ->countdel.
OUTPUT countdel

END.

MAKE MULT_ADD:mult_add, #the code for the convolver#

[4]DF1(t_scratch)-pdel, COUNT_2:count.

now the state machines to control the convolver# #First the and gates#

LET

#starts row counter 1 cycle after frame start# reset_col=DF1[t_reset](ck,reset),

#makes up for the pipeline delay in MULT#

#IIILATENCY DEOENDENTII#

#flag when col_count≖0,1,2,col_length,etc#

col_control=col_flag.

andsel=(CASE direction

one:pass, two:zero forward: CASE count

ESAC, Inverse: CASE count

OF one:zero, two:pass

ESAC

CASE col_control OF count_0:zero

ELSE pasa ESAC,

forward: CASE col_control CASE direction

OF count_0zero ELSE pass ESAC,

inverse: pass

#now the add/sub control for the convolver adders# OF one:(add,add,add,sub), two: (add,sub,add,add) ESAC, addsel= CASE count

#now the mux control#

centermuxsel=

CASE direction

forward: CASE count

OF one:(left,right), two:(right,left) ESAC,

inverse:CASE count

one:(right,left), two:(left,right)

ь Б

ESAC

ESAC.

#the addmuxsel signal#

muxandsel=

CASE direction

forward:(andsel[2],pass,andsel[2]), inverse:(pass,andsel[2], CASE col_control

count_1:zero

ELSE pass ESAC)

forward:(uno,

9

ESAC,

CASE direction

muxsel=

CASE col_control OF count_0:dos,

count_carry:tres

ELSE uno ESAC, CASE col_control OF count_0.tres,

count_carry:quatro

ELSE dos

ESAC),

inverse: CASE col_control OF count_0:dos, count_1:quatro, count_tm1 tres

ELSE dos ESAC,

CASE col_control OF count_0.tres,

count_carry:dos

ELSE uno ESAC,

Oun Ou

ESAC.

#join the control signals to the mult_add block#

JOIN (ck,reset_col) ->count, #set up the col counters #

(ck,reset,in,andsel,centermuxsel,muxsel,muxandsel,addsel,direction,pdel)->muit_add.

FOR INT j=1..4 JOIN

(ck,mult_add[]] ->pdel[]].

#pipeline delay for muft-add unit#

#ACTEL HACK#

gh_select=CASE direction
OF inverse: CASE c E

CASE count

one: left,

two: right
ESAC,
CASE count
OF one:right,
two:left forward:

ESAC

ESAC,

gh_out = MUX_2(t_scratch)(pde[[4],DF1(t_scratch)(ck, pde[[1]),gh_select),

rb_select= CASE direction OF inverse:CASE

inverse: CASE col_control

(count 2 | count 3):shift3

ELSE shift4 ESAC, shift5

forward: shift5. ESAC. OUTPUT ROUND_BITS(gh_out,rb_select)

#some string macros# MAC EQ_US = (STRING[INT n]bit: a b)

bool: BIOP EQ_US.

#ACTEL 8 bit comparitor macro# FN ICMP8 = (STRING[8]bit: a b)

bool: EQ_US(8)(a,b).

MAC COUNT_SYNC(INT n) = (bool ck,t_reset: reset,bool: en)

#The basic toggle flip-flop plus and gate for a synchronous counter #input t is the toggle ,outputs are q and tc (toggle for next counter# #stage	ounter # unter# #
et bool: tog)	.> [2]bool:
BEGIN MAKE DFF_NO_LOAD[bool];dlat, XOR :xor, AND :and.	
JOIN (ck,reset,xor,f)->dlat, (dat,tog) ->and, (tog,dlat) ->xor. OUTPUT (dlat,and)	
# # The n-bit macro counter generator, en is the enable, the outputs #	touts #

(flag,t_col):BIOP TRANSFORM_US.

MAKE COUNT_SYNC[xsize]:count,

BOOL_STRING(xsize):b_s.

JOIN (ck,reset,t) ->count,

#count always enabled#

count[1]->b_8. OUTPUT (S_TO_C b_s)[2] END. #a mod 2^ysize counter# MAC MOD2_COUNTER_ROW = (boot.ck,t_reset:reset,boot.en)

BEGIN

MAC S_TO_C = (STRING(xsize)bit.in)

MAC S_TO_R = (STRING[ysize]bit:in)

(t_row):

MAKE COUNT_SYNC[ysize]:count, BOOL_STRING[ysize]:b.s.

(flag,1_row):BIOP TRANSFORM_US.

۸

JOIN (ck,reset,en) ->count, count(1) ->b_8. OUTPUT (S_TO_R b_s)[2]

MAC BASE_COUNTER_COL = (boot.ck,1_reset.reset,STRING|xsize|bit:octave_cnt_length) #the basic mod col_length counter, to be synthesised#

(1_col,1_count_control):

MAC C_TO_S = (1_col: in)

(flag,STRING(xeize|bit): BIOP TRANSFORM_US. MAC FINAL_COUNT = (t_col.in,STRING[xeize]bit:octave_cnt_length]

.> t_count_control:

BEGIN

LET in_us = (C_TO_S in)[2],

Isb=in_us(xsize). #OUTPUT CASE EQ_US(in_us[1..xsize-1],octave_cnt_length[1..xsize-1]) the msb's are the same#

BEGIN

```
OUTPUT CASE ICMP8(in_us[1..xsize-1],octave_cnt_length[1..xsize-1]) #the msb's are the same#
                                                                                                #count is even so must be length-1#
                                                             #count odd, so must be length#
                               #so check the lsb#
                                                                 OF b'1:count_carry,
                                                                                                  b'0:count Im1
                               OF t: CASE Isb
```

ESAC

ELSE count_rst

ESAC

END.

MOD2_COUNTER_COL:mod2_count, FINAL_COUNT:final_count. MAKE

->final_count, [mod2_count,octave_cnt_length] NO NO

ck, CASE reset

OF nat: nat

#system reset or delayed carryout reset#

ELSECASE DFF_NO_LOAD[L_count_control](ck,reset,final_count,count_0) #latch to avoid gitches#

OF count_carry:rst

ELSE no_rst ESAC

->mod2_count.

OUTPUT (mod2_count,final_count) ESAC)

FN COL_COUNT_ST = (bool:ck,t_reset:reset,STRING[xsize]bit:octave_cnt_length)

(1_col,t_count_control):

#count value, and flag for count=0,1,2,col_length-1, col_length#

BEGIN

MAKE BASE_COUNTER_COL.base_col

count_control = CASE reset E

```
OF rst:count_0
ELSE CASE base_col[1]
OF col0:count_0,
col/1:count_1,
col/2:count_2,
col/3:count_3
```

ELSE base_co[[2] ESAC

ESAC. JOIN (ck, reset, octave_cnt_length) ->base_cd.

(base_cof(1),count_control)

END

OUTPUT

#the basic mod row_length counter, to be synthesised# MAC BASE_COUNTER_ROW = (bool:ck,t_reset:reset,bool:en,STRING[yeize]bit:octave_cnt_length,t_count_control:col_carry)

(t_row,t_count_control):

BEGIN

MAC R_TO_S = (I_row:in)

flag, STRING(yeizejbit): BIOP TRANSFORM_US.

MAC FINAL_COUNT = (I_row:in,STRING[ysize]bit:octave_cnt_length)

t_count_control:

BEGIN

LET in_us = (R_TO_S in)[2], sb=in us[ysize].

#OUTPUT CASE EQ_US(in_us[1..ysize-1],octave_cnt_length[1..ysize-1]) the msb's are the same#

OUTPUT CASE ICMP8(in_us[1..ysize-1],nctave_cnt_length[1..ysize-1]) #the msb's are the same# #so check the lsb# OF b'1:count_carry. OF 1: CASE lab

#count odd, so must be length# b'0:count lm1

#count is even so must be langth-1#

ESAC

ELSE count_rist

ESAC

END.

MAKE MOD2_COUNTER_ROW:mod2_count,

FINAL_COUNT:final_count.

#need to delay the reset at end of count signal till end of final row#

#WAS DFF WITH reset#

LET count_reset = DF1 (t_reset) (ck, CASE (final_count, col_carry) #last row/last col#

OF (count_carry,count_carry):ret #fatch to avoid gitiches#

ELSE no_rst

ESAC).

JOIN (mod2_count,octave_cnt_length) ->final_count,

#system reset or delayed carryout reset# (ck,CASE reset

OF nst: nst

->mod2_count. ELSE count_reset ESAC,en)

OUTPUT (mod2_count,final_count)

FN ROW_COUNT_CARRY_ST = (bool:ck,t_reset.reset,STRING[ysize]bit:octave_cnt_length,t_count_control:col_carry)

(t_row,t_count_control):

```
MAKE BASE_COUNTER_ROW base_row.
                                                           ELSE CASE base_row[1]
                                                                              row/0:count_0,
                                                                                              row/1:count_1
                           count_control = CASE reset
                                              OF rst:count_0
BEGIN
                               LET
```

ELSE base_row[2] ESAC

row/2:count_2, row/3:count_3

ESAC.

ESAC, octave_cnt_length,col_carry) (base_row[1],count_control) OF count_carry:t JOIN (dk,reset,CASE col_carry ELSE OUTPUT

#when ext & csl are both low latch the setup params from the nubus(active low), as follows# #the discrete wavelet transform chip/ multi-octave/2d transform with edge compensation# select function# fadi[1..4]

oad max_octaves, luminance/colour, forward/inversebar# bad yimage# 800 900 100

bad ximage# jump table values# 0010

bad 3ximage+3# bad 7ximage+7# load ximage+1# <u>6</u> 0100 0101

load base v addr# load base u addr# 0110 <u>=</u>

#adi[23] Luminance/crominancebar active low, 1 is luminance, 0 is colour# #adi[24] lonward/inversebar active low, 1 is forward, 0 is inverse# max_octaves# #ad[[21..22]

data (bit 24 lsb)# #adl[5..24]

FN ST_OCT = (STRING[2]bit:st)

(flag,t_octave): BIOP TRANSFORM_US.

FN OCT_ST = (t_octave:st)

(flag,STRING[2]bit):BIOP TRANSFORM_US.

FN DWT = (bool.ck_in,t_reset.reset_in, t_input.in_in,bcol.extwritel_in csl_in, STRING[24]bit.adl, t_input:sparc_mem_in, [4]t_scratch:pdel_in)

(t input#out IDWT data#,[3]t_bad#valid out IDWT data,y,u,v#,

Leparcport#sparc_data_addr, etc#, [3] toad#valid in DWT data y,u,v#,

[_memport#pdel_data_out#):

MAKE CONV_2D:conv_2d, ADDR_GEN_NOSCRATCH:addr_gen,

#active low clock &enable latches#

DLE1D:channel_factor_st,
DLE1D:dir,
[9]DLE1D:coi_length_s,
[9]DLE1D:xoi_length_s,
[1]DLE1D:x_p_1,
[1]DLE1D:x_p_1,
[1]DLE1D:x_p_1,
[1]DLE1D:x_p_1,
[1]DLE1D:base_u,
[1]DLE1D:base_u,
[1]DLE1D:base_u,
[1]DLE1D:base_u,
[1]DLE1D:base_u,
[1]DLE1D:base_u,
[1]DLE1D:base_u,
[1]DLE1D:coder#
DEC3X8A

#the octave control#
DFF_INIT(I_cdannel]: channel,
JKFF:row_carry_ff,

[2]DLE1D:max_octave_st

INBUF[STRING[24]bit] adl_out, CLKBUF:ck, INBUF[bool]:extwritel csl,

#bad#

INBUF(t_reset):reset, INBUF(t_input):in sparc_mem, INBUF([4]t_scratch):pdet,

OBHS[[3]t_load]:out2 out3, OBHS(t_sparcport]:out4.

OBHS(t_input):out1,

OBHS(t_sparcport):out4, OBHS(t_memport):out5.

UBHIST MEMPORTS. US. THE WITE CONTROL TO MATCH THE CONTROL OF CONTROL OF CONTROL OF CONTROL PARENCY.

#set up the control params#

max_oct = (ST_OCT BOOL_STRING(2)max_octave_st)[2].

channel_factor= CAST(t_channel_factor)channel_factor_st,

col_length = BOOL_STRING(9) col_length_s,

row_length = BOOL_STRING[9] row_length_s.

direction =CASE dir OF f:forward,

ESAC,

#set up the octave parame#

convcol_row= conv_2d[3],

convcol_col=conv_2d[4],

convrow_col=conv_2d[5],

#signals that conv_col, for forward, or conv_row, for inverse, has finished that octave#

#and selects the next octave value and the sub-image sizes#

octave_finished =CASE direction

OF toward:CASE (row_carry_fl,convcol_row,convcol_col)

OF (t,count_2,count_2).write #row then col, gives write latency#

ELSE read

ESAC,

inverse:CASE (row_carry_ff,convcol_row,convrow_col)
OF (t,count_2,count_3):write #extra row as col then row#

ELSE read ESAC

ESAC, #max octaves for ulv#

oct/1:oct/0, oct/2:oct/1, oct/3:oct/2 = CASE max_oct ESAC, y_done =CASE (channel,(OCT_ST octave)[2] EO_US CASE direction OF forward:CAST(STRING [2]bit]max_octave_st,

inverse:b*00*

ESAC)

(y,t);t ELSE 1 ESAC,

uv_done = CASE (channel,(OCT_ST octave)[2] EQ_US CASE direction
OF torward:(OCT_ST max_oct_1)[2],
inverse.b*00*

ESAC)

OF (u|v,1)1 ELSE f ESAC,

(SEQ nexta

new_channel:=channel; CASE direction VAR new_oct:=octave,

forward:(CASE octave OF oct/0:new

9

oct/0:new_oct:=oct/1, oct/1:new_oct:=oct/2,

oct/2:new_oct:=oct/3

ESAC;

```
CASE (y_done,uv_done)
OF (t,bool)|(bool,t):new_oct:=oct/0
ELSE
ESAC
),
inverse:(CASE octave
```

r ocoo.case charine iactol #watch to consul# OF luminance:new_oct:=max_oct_1 ELSE new_oct:=max_oct_1 ESAC

ESAC, u:CASE octave OF oct/0_new_oct:=max_oct_

ELSE ESAC, v:CASE octave

OF oct/0 new_oct:=max_oct ELSE ESAC

#move to y#

SAC

```
CASE channel factor

OF tuminance:new_channel:=y,
color: (CASE (channel,y_done))

OF (y,t):new_channel:=u
ELSE
ESAC;
CASE (channel,uv_done)

OF (u,t):new_channel:=y
ELSE
ESAC)

ELSE
ESAC)

ELSE
ESAC)
```

ESAC;

```
octave_row_length =MUX_4{STRING [ysize]bit]{row_length,b^0^CONC row_length[1..ystze-1], b^00^CONC row_length[1..ystze-2],
                                                                                                                                                                                                                                                                                                                                                       b*000* CONC row_length[1..ysize-3],octave_sel], octave_col_length = MUX_4{STRING [xsize]bit]{col_length,b*0* CONC col_length[1..xsize-1],
octave_sel = CASE (octave,channel) #the block size divides by 2 every octave#
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            b"000" CONC col_length[1..xsize-3],octave_sel],
                                              #the ulv image starts 1/4 size#
                                                                                                                                                                                                                                                                                                                                                                                                                                                b*00* CONC col_length[1..xsize-2],
                                                                                                                                                                           oct/3.y)|(oct/2,u|v):quatro
ESAC,
                                                                                     oct/1,y))(oct/0,u|v):doe,
                                                                                                                                  oct/2,y)[(oct/1,u|v):tres,
                                              (oct/0,y):uno,
```

OUTPUT (new_od,new_charmel)

#load next octave, either on system reset, or write finished# CASE reset OF rst.write load_octave=

ESAC,

ELSE octave_finished

#reset the convolvers at the end of an octave, ready for the next octave#

#cant glitch as reset&octave_finished dont change at similar times# #latch pulse to clean it, note 2 reset pulses at frame start#

conv_reset = CASE reset

OF rat:rat

ELSE CASE DFF_NO_LOAD(1_load)(ck,reset, octave_finished,read)

OF write:rst

ELSE no_rst

ESAC

#latch control data off nubus, latch control is active low# CASE (extwritel,csl)

#write addresses# sparc_w=addr_gen[1][2][1].

input_mux=addr_gen[1][1], #input_mux#

#read addresses# sparc_r=addr_gen[1][2][2],

sparc_rw = addr_gen[1][2][3]

```
(y,oct/0,read):(read,write,write)
                                                                                                                                                                                                                                                                 u,oct/0,read):(write,read,write)
                                                                                                                                                                                                                                                                                   (v,oct/0,read):(write,write,read)
                                                                                                                                                                                                                      forward:CASE (channel,octave,addr_gen[3]
                  (inverse,oct/0).CASE (channel,addr_gen[2])
                                      (y,write):(write,read,read),
                                                        (u,write):(read,write,read)
                                                                              (v,write):(read,read,write)
                                                                                                                                                                                                                                                                                                     ELSE (write, write, write)
                                                                                                                                      (forward,oct/0):(read,read,read)
                                                                                               ELSE (read,read,read)
ESAC,
                                                                                                                                                                                                                                                                                                                                              inverse:(write,write,write)
CASE (direction,octave)
OF (inverse,oct/0):C
                                                                                                                                                          ELSE (read,read,read)
                                                                                                                                                                             ESAC,
CASE direction
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  ->sparc_mem,
                                                                                                                                                                                                                                                                                                                                                                                                                                                                    ->extwritel,
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          ->adl_out,
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       ->C8i,
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               sparc_mem_in
pdel_in
                                                                                                                                                                                                                                                                                                                                                                                                                                               reset_in->reset
   inverse_out =
                                                                                                                                                                                                   forward in =
                                                                                                                                                                                                                                                                                                                                                                                                                                                                  extwritel in
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            .E.
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        #ont bads#
                                                                                                                                                                                                                                                                                                                                                                                                         #in pads#
                                                                                                                                                                                                                                                                                                                                                                                    N<sub>O</sub>
```

#active low outs# (CAST[bool]adi[4],CAST[bool]adi[3],CAST[bool]adi[2]) ->decodel, #the control section#

->out2, ->out3,

inverse out conv_2d[1]

lorward in

->out4,

conv_2d[2] ->oul5, addr_gen[1][2]

->out1,

->max_octave_st[1], (gl,decode[[1],BIT_BOOLadi_out[21]) (gl,decode[[1],BIT_BOOLadi_out[22])

->max_octave_st[2]

->channel_factor_st,

(gl,decodel[1],BIT_BOOLadl_out[23]) (gl,decodel[1],BIT_BOOLadl_out[24])

->row_length_c[] ->col_length_s[] (gi,decode|[2],BIT_BOOLadi_out[15+]]) (gi,decode|[3],BIT_BOOLadi_out[15+]]) (gi,decode|[4],BIT_BOOLadi_out[15+]])

FOR INT := 1..9 JOIN

->×_P_4.×-

->x3_p_1[]]. [gi,decodel[5],BIT_BOOLadl_out[13+j])

FOR INT F-1..11 JOIN

FOR INT JE1..12 JOIN

FOR INT |=1..19 JOIN

->x7_p_1[] gi,decode[[6],BIT_BOOLadi_out[12+]])

->base_u[j], ->base_v[j].

(gi,decodel[7],BIT_BOOLadi_out[5+]]) (gi,decodel[8],BIT_BOOLadi_out[5+]])

#sets a flag when row counter moves onto next frame#

(ck,conv_reset,CASE convcol_row OF count_carry.t

->row_cairy_ff, ESAC,()

on initial reset must load with starting octave value which depends on direction and channel# #load the new octave, after the current octave has finished writing# (ck,no_rst,load_octave, CASE reset

OF no_rstriext[1] ELSE CASE (direction, channel) #initial octave#

OF (forward,t_channel):oct/0,

(inverse_y):max_oct,

(inverse,uly):max_oct_1

ESAC

->octave, #next octave# ESAC,oct/0)

(ck,no_rst,load_octave, CASE reset

no_ret:next[2]

ELSE y ESAC,y)

->channel, #next channel#

(ck,reset,MUX_2(t_input){in,sparc_mem,CASE input_mux #input_mux#

OF dwt_in:left,

sparc_in:right ESAC)

->conv_2d, direction, pdel, conv_reset, addr_gen[4], addr_gen[5])

->addr_gen. (ck,reset,direction,channet,BOOL_STRING(9)x_p_1,BOOL_STRING(11)x3_p_1, BOOL_STRING(12)x7_p_1,octave_row_lengtin, octave_col_length,conv_reset,octave,y_done,uv_done,octave_finished,BOOL_STRING(19)base_u, BOOL_STRING(19)base_v)

(out1, out2, out3, out4, out5) OUTPUT

FN DWT_TEST = (bool:ck_in,t_reset:reset_in, t_input:in_in,bool:extwritel_in csl_in,t_sparc_addr:reg_sel value)

```
(t_input,[3]t_load,[3]t_load):
```

۸

FN SPARC_MEM = (1_input:in,t_sparc_addr.wr_addr,t_sparc_addr.rd_addr,t_bad:rw_sparc#,t_cs:cs#)

t_input: RAM(input/0).

MAKE DWT:dwt, SPARC_MEM:sparc_mem,

LINE DELAY(t_scratch):line_delay.

sparc_port=dwt[4], data_out=dwf1], E

line_delay_port = dwt[5].

(ck_in,reset_in,in_in,extwritel_in, cst_in,(SPA_S reg_set)[2][16..19]CONC b*1* CONC(NOT_B (SPA_S value)[2]), sparc_mem,line_delay) N S

->sparc_mem, (data_out,sparc_port[1].sparc_port[2].sparc_port[3]#,sparc_port[4]#)

(line_delay_port[1],line_delay_port[2],line_delay_port[3],write) ->line_delay

OUTPUT

some basic macros for the convolver, assume these will#

the synthesised into leaf cells#

#the actel MX4 mux cell#

FN NOT = (boof.in)

bool:CASE in OF t:1,ft ESAC.

```
MAC MX_4[TYPE ty]=(ty.in1 in2 in3 in4, [2]boot3e8]

CASE sel
OF (t,f).in2,
(t,f).in3,
(t,f).in4

ESAC.

#the actel GMX4 mux cell#
MAC GMX4[TYPE ty]=(ty.in1 in2 in3 in4, [2]boot3e8]

Att.
CASE sel
OF (t,f).in1,
(t,f).in2,
(t,f).in3,
(t,f).in4

ESAC.

MAC MXT[TYPE ty]=(ty.a.b.c.d, boot3e0a sob s1)

CASE st

CASE soa
OF tb

ELSE a

ESAC,
t: CASE soa
OF tb

ELSE a

ESAC,
t: CASE sob
```

ESAC. MAC ENCODE4_2 = (t_mux4:in)

MAC ENCODES_2 = (L_mux3.in)

ESAC.

FN DEC3X8A = (bool:a b c)

ESAC.

MAC MUX_2(TYPE t)=(tint in2, t_mux:sel)

CASE sel OF left

ESAC.

MAC MUX_3(TYPE I)=(tin1 in2 in3, Lmux3xel) MX_4[i](in1,in2,in3,in1,ENCODE3_2 sel). COM MAC MUX_4[TYPE t]=(t:in1 in2 in3 in4, 1_mux4.xel)

unoin1,

dos in 2, tres in 3, quatro in 4

ESAC. MOC

```
MAC MUX_4(TYPE t]=(t:in1 in2 in3 in4, t_mux4:sel)
                                                                                                   MX_4[t](in1,in2,in3,in4,ENCODE4_2 sel).
```

FN AND2 = (bool:a b) bool:BIOP AND. MAC GNAND2 = (bool:a b)

bool:NOT AND2(a,b).

MAC AND_2 = (L_scratch:in, t_and:sel)

t scratch: BEGIN LET in s

in_s = (l_TO_S(scratch_explin)[2], sel_s = CAST[bool]sel.

OUTPUT (S_TO_I[scratch_exp]BOOL_STRING[scratch_exp] ([INT j=1..scratch_exp]AND2(BIT_BOOL in_s[j],sel_s)))[2]

FN XOR = (bool: a b)

pool:

CASE (a,b)

MAC XOR_B(INT n) = (STRING[n]bit:a b)

STRING[n]bit: BIOP XOR.

MAC NOT_B = (STRING[INT n]bit:a)

STRING[n]bit:BIOP NOT.

MAC XNOR_B = (STRING[INT njbit:a b)

STRING[n]bit: NOT_B XOR_B[n](a,b).

FN AND = (bool: a b)

CASE (a,b) <u>500</u>

OF (I,t):1 ELSE ! ESAC.

MAC DEL[TYPE I] - (I)

t:DELAY(?1,1).

#a general dif same as DFF_NO_LOAD# MAC DFF {TYPE t}=(bool:ck,t_reset.reset,t.in init_value)

BEGIN

MAKE DEL(I):del.
JOIN in->del.
OUTPUT CASE reset
OF ret:init_value
ELSE del

END

#a general dff# MAC DF1 (TYPE t)=(bool:ck,t.in)

t: Begin Make Del(1):del

JOIN in->del. OUTPUT del #a general latch# MAC DL1 {TYPE ty}=(bool:ck,ty:in)

BEGIN MAKE DEL(ty):del JOIN CASE ck OF t:in

OF Lin ELSE del ESAC ->del

ELSE del ESAC

ESAC END. #a general d tatch# MAC LATCH [TYPE t]=(boot:ck,t_bad:bad,t:in)

BEGIN MAKE DELI¶:del.

LET out=CASE load
OF write:in

ELSE del ESAC.

JOIN out->del.
OUTPUT out

#an ACTEL D LATCH# MAC DLE1D = (bool:ckl loadl,bool:in) NOT LATCH(bool) (NOT ckt, CASE load!
OF !:write

bool:#dn#

ELSE read ESAC, in). MAC PDF1{TYPE t,INT n} = (bool:ck,t_reset:reset,t:in initial_vatue)
->

IF n=0 THEN DFF(t)(ck,reset.in,initial_value)

```
PDF1(t,n-1)(ck,reset,DFF(t)(ck,reset, in,initial_value),initial_value)
 ELSE
FI.
```

#a muxed input dff# MAC DFM [TYPE ty]=(bool.ck,ty.a b,bool.s)

BEGIN

MAKE DEL(ty):del.

JOIN CASE 8 OF f.a,

ع

ESAC

OUTPUT del

END.

#a resetable DFF, init value is input parameter#

MAC DFF_INIT[TYPE t]=(bool:ck,t_reset:reset,t_load:bad,tin init_value)

BEGIN

LET out=CASE (load,reset) MAKE DEL(I):del.

(read,rst):init_value OF (write,t_reset):in,

ELSE del ESAC. JOIN out->del.

OF ret.init_value ELSE del **OUTPUT CASE reset**

```
E:SAC
END.
```

```
#a resetable JKFF, k input is active low#
FN JKFF=(bocl:ck,t_reset:reset,bootj k)
```

pool:

BEGIN

MAKE DEL(bool):del. LET out=CASE (j.k,resel)

OF (I,t.no_rst).t, (I,t.st).f, (I,f.st).t,

(f,f,no_rst):f, (f,f,no_rst):del, (f,f,no_rst):NOT del

(t,f,no_rst):N ESAC.

OUTPUT CASE reset OF rst:f

ELSE del

ENO.

#a dif resetable non- loadable dif#
MAC DFF_NO_LOAD(TYPE t)=(bool:ck,t_reset:reset,t:in init_value)

BEGIN

MAKE DEL(I):del. JOIN in->del.

JOIN out->del.

OF rst.init_value ELSE del ESAC END. MAC PDEL(TYPE t,INT n) = (t.in)

OUTPUT CASE reset

ind of Declinite that if = (1.20)

If n=0 THEN DEL(t)in

ELSE PDEL(t,n-1) DEL(t) in
F1.

MAC MEM_CONTROL_NOSCRATCH = (bool:ck,t_reset:reset;t_direction:direction,t_channel:channel,t_octave:octave, #the mem control unit for the DWT chip, outputs the memport values for the spare, and dwt# 1_sparc_addr:sparc_addr_w sparc_addr_r,1_load.zero_hh) #inputs datain from these 2 ports and muxs it to the 2d convolver.#

(t_input_mux,t_sparcport,t_dwtport#dwt#):

ņ

BEGIN
#the comb. logic for the control of the I/o ports of the chip#
LET ports = (SEQ

VAR #defaults, so ? doesnt kill previous mem vatue#
rw_sparc:=read,
rw_dwt:=read,
cs_dwt:=no_select,
input_mux:=sparc_in;

#rw_sparc=write when ck=1 and zero_hh=write, otherwise = read# rw_sparc:= CAST(!_load)GNAND2(NOT CAST{bool}zero_hh,ck);

ESAC;

#mux the sparc addr on clock#

sparc_addr = GMX4(t_sparc_addr)(sparc_r,sparc_w,sparc_w,ck,f);#

OUTPUT (input_mux, (sparc_addr_w,sparc_addr_r,rw_sparc), #sparc port#

(rw_dwf,cs_dwt)

#dwf port#

(m__m'm__m)

). OUTPUT ports END. # the basic 1d convolver without the control unit#

MAC MULT_ADD = (bool:ck,t_reset: reset,t_input:in, [3]t_and:andsel, [2]t_mux:centermuxsel_[3]t_mux4:muxsel, (3)t_and:muxandsel[4]t_add:addsel, t_direction:direction[4]t_scratch:pdel)

[4]t_scratch: #pdel are the outputs from the line delays#

BEGIN

MAKE MULTIPLIER:mult,
[4]ADD_SUB: add.
#the multiplier outputs#
LET x3=mult[1],
x1=mult[2],
x1=mult[3],
x2=mult[4],
x2=mult[5],
x8=mult[6],
x30=mult[7],

#the mux outputs# mux1=MUX_4[t_scratch](x11,x5,x8,x2,muxsel[1]), mux2=MUX_4(I_scratch)(x19,x30,x8,scratch/0,muxse{{2}}),

mux3=MUX_4{[_scratch](x11,x5,x8,x2,muxse[[3]),

centermux=(MUX_2(L_scratch)(pdel[1],pdel[3],centermuxsel[1]), MUX_2(L_scratch)(pdel[2],pdel[4],centermuxsel[2])),

the AND gates zero the adder inputs every 2nd row# #the and gate outputs#

and1=AND_2(pdel[2],andsel[1]), and2=AND_2(pdel[3],andsel[1]), and3=AND_2(centermun[1],andsel[2]), and4=AND_2(centermun[2],andsel[3]),

add1in=AND_2(mux1,muxandsel[1]).

add3in=AND_2(mux3,muxandsel[2]). add4in=AND_2(x3,muxandsel[3]).

and4,add3in,addset[3]) ->add[3], and1,add1in,addse[[1]] ->add[1] and2,add4in,addse[4]) ->add[4] (and3,mux2,addee[[2]) ->mult. NS S

OUTPUT add

the basic multiplier unit of the convolver

MAC MULTIPLIER_ST = (t_input:in)

[7]L_scratch: #x3,x5,x11,x19,x2,x8,x30#

MAC INPUT_TO_S(INT n) = (t_input: in)

BEGIN

Ņ

(flag,STRING[n]bit): BIOP TRANSFORM_S.

#the multiplier outputs, fast adder code commented out# In_s= (INPUT_TO_S[input_exp]in)[2], 回

x2=in_s CONC b'o',

x8=in_s CONC b'000",

x3 = ADD_S_ACTEL(in_s, x2,b'1), x5 = ADD_S_ACTEL(in_s,in_s CONC b'00",b'1), x11 = ADD_S_ACTEL(x3,x8,b'1), x19 = ADD_S_ACTEL(x3,in_s CONC b'0000",b'1),

x30=ADD_S_ACTEL(x11,x19,b1).

OUTPUT ((S_TO_l[input_exp+2] x3][2],(S_TO_l[input_exp+3] x5][2],(S_TO_l[input_exp+4] x11)[2],
(S_TO_l[input_exp+5] x19][2],(S_TO_l[input_exp+1] x2)[2],(S_TO_l[input_exp+3] x8][2],
(S_TO_l[input_exp+6] x30][2]) LET subsignal = (x2,x8, x3,x5,x11,x19,x30)

ב

MAC INBUF(TYPE I) = (t:pad)

t:#y#pad.

MAC OBHS(TYPE 1) = (1:d)

t:#pad#d

FN CLKBUF = (bool:pad)

.

#MAC SHIFT(INT p) = (STRING[scratch_exp]bit) ->STRING[scratch_exp+p]bit:BIOP SR_S[p].# bool:pad.

MAC ADD_S = (STRING[INT m]bit, STRING[INT n]bit)

STRING[IF m>=n THEN m+1 ELSE n+1 FI]bit: BIOP PLUS_S.

MAC INV[INT m] =(STRING[m]bit:a)

STRING[m]bit:BIOP NOT.

MAC NEG_S = (STRING(INT n]bit)

^

BIOP NEGATE_S. STRING[n+1]bit:

MAC ADD_US = (STRING[INT m]bit,STRING[INT n]bit)

STRING[IF m> n THEN m+1 ELSE n+1 FI]bit: BIOP PLUS US.

MAC CARRY= (L_add:in)

STRING[1]bit: CASE in

OF add:b"0",

sub:b.1

ESAC.

#actel adder macros#

#an emulation of a fast ACTEL 16 bit adder with active low carrys FN FADD16 = (STRING[scratch_explbit: a b,STRING[1]bit:cinb)

(STRING[scratch_exp]bit,STRING[1]bit);

BEGIN

b_c = b CONC INV(1) cirb, a_c =a CONC INV(1)cinb,

out = ADD_S(a_c,b_c).
OUTPUT(out[2..scratch_exp+1],INV[1] B_TO_S out[1])

#actel 1 bit full adder with active low cin and cout# MAC FA1B - (bit: ain bin cinb)

(bit,bit):#cob,s# BEGIN

a_c = B_TO_S ain CONC INV(1)B_TO_S cinb, b_c = B_TO_S bin CONC INV(1)B_TO_S cinb,

out = ADD_US(a_c,b_c).
OUTPUT(CAST(bit) INV(1) B_TO_S out[1], out[2])

#the actel version of the ADD BIOP's#

MAC ADD_US_ACTEL = (STRING[INT m]bit:ain,STRING[INT n]bit:bin,bit:cinb)

STRING(IF m>=n THEN m+1 ELSE n+1 FIJbit:

BEGIN

MAKE (IF m>=n THEN m ELSE n FIJFA1B:sum.

#unsigned nos so extend by 0#

LET a_c = IF m>=n THEN ain ELSE ZERO(n-m)b"0" CONC ain FI,

b c = IF n>=m THEN bin ELSE ZERO(m-n)b'0" CONC bin FI.

LET subsignal = sum.

(a_qiF m>=n THEN m ELSE n Fij,b_qiF m>=n THEN m ELSE n Fij,cinb) ->sum(iF m>=n THEN m ELSE n Fij NO NO

FOR INT j=1..(IF m>=n THEN m ELSE n FI) -1

JOIN (a_d(IF m>=n THEN m ELSE n FI) -jj,b_d(IF m>=n THEN m ELSE n FI) -jj, sum{(IF m>=n THEN m ELSE n FI) -j+1 || 1) >sum{(IF m>=n THEN m ELSE n FI) -jj.

CASTISTRINGIIF m>=n THEN m+1 ELSE n+1 Filbit OUTPUT

(INV[1] B_TO_S sum[1][1] CONC

->sum((F

CAST[STRING[IF m>=n THEN m ELSE n FIJbil] [INT]=1..IF m>=n THEN m ELSE n FI] sum[j][2])

MAC ADD S_ACTEL = (STRING(INT m/bit:ain,STRING(INT n/bit:bin,bit:cinb)

STRING(IF m>=n THEN m+1 ELSE n+1 FIJbit:

BEGIN

MAKE (IF m>=n THEN m ELSE n FIJFA18:sum.

#signed nos so sign extend #

LET a_c = IF m>=n THEN an ELSE ALL_SAME(n-m)B_TO_S ain[1] CONC ain FI, b_c = IF n>=m THEN bin ELSE ALL_SAME(m-n)B_TO_S bin[1] CONC bin FI.

LET subsignal = sum.

#lab#

JOIN (a_c(if m>=n THEN m ELSE n Fij,b_c(if m>=n THEN m ELSE n Fij,cinb) ->sum(if m>=n THEN m ELSE n Fij.

FOR INT j=1..(IF m>=n THEN m ELSE n FI) -1

JOIN (a_c[(IF m>=n THEN m ELSE n FI) -j],b_c[(IF m>=n THEN m ELSE n FI) -j], sum[(IF m>=n THEN m ELSE n FI) -j+1 [[1]) m>=n THEN m ELSE n FI) -jj-1 [[1].

OUTPUT CAST(STRING(IF m>=n THEN m+1 ELSE n+1 Fijbit)
(INV(1) B_TO_S sum[1][1] CONC
CAST(STRING(IF m>=n THEN m ELSE n Fijbit) [INT j=1...|F m>=n THEN m ELSE n Fij sum[][2])

END.

FN ROUND_BITS = (t_scratch:in,t_round: select)

BEGIN

```
#THIS ASSUMES THAT THE INPUT_EXP=10!!!!#

#select chooses a round factor of 3, 4,5#

#the Isb is the right hand of the string,#

#the index 1 of the string is the left hand end, &is the msb#

#so on add ops bit 1 is the carryout#

LET s1= (I_TO_S(scratch_exp]in){2}.
```

msb= B_TO_S st[1],

selector = CASE select #case conversion for MUX_3#

OF shift31, shift4.c,

shiff5:r

ESAC,

#needs to be a 16 bit output for the adder#
shift = MUX_3{STRING{scratch_exp]bit}(

msb CONC msb CONC msb CONC st[1..scratch_exp-3],
msb CONC msb CONC msb CONC msb CONC st[1..(scratch_exp-4)],
msb CONC msb CONC msb CONC msb CONC msb CONC st[1..scratch_exp-5],
selector

#the carry to round, 1/2 value is rounded towards 0#

cs = CASE select

OF shift4: CASE msb

OF b"1":s1[scratch_exp-3], #neg no.# b"0": CASE s1[scratch_exp-3..scratch_exp]

OF b-1000": b'0 #round down on 1/2 value#

ELSE s1[scratch_exp-3]

ESAC

ESAC, shift3: CASE msb

#neg no.# OF b'1":s1[scratch_exp-2],

b.0.: CASE s1[scratch_exp-2..scratch_exp]

#round down on 1/2 value# OF b*100": b'0

ELSE st[scratch_exp-2]

ESAC

ESAC,

shift5: CASE msb

#neg no.# OF b"1":s1[scratch_exp-4],

b.0: CASE st[scratch_exp-4..scratch_exp]

Fround down on 1/2 value# OF b'10000": b'0

ELSE s1[scratch_exp-4]

ESAC

ESAC

sum17 =ADD_US_ACTEL(B_TO_S cs, shift,b'1), sum = sum17[2..scratch_exp+1],

#bit 1 is carry out, gives 16 bit sum#

subsignal=(cs,sum),

#ACTEL HACK#

soa = CASE sum[1]

OF 51:1, #saturate to -512# b'0: f #saturate to 512#

ESAC,

ss1 = CASE selector

CASE sum[4..7] #these are the 5 msb's form the 13 bit word# OF (b*1111* | b*0000*): t#value in range#

ELSE 1 ESAC, CASE sum[5..7]#these are the 3 msb's from the 12 bit word left after# # taking out the 4 sign extension bits# ပ

#value in range# OF (b-111-16-000"): 1

ESAC,

CASE sum[6..7] #these are the 2 msb's from the 11 bit word# OF (b*11* | b*00*): I #value in range# ت

ELSE 1

ESAC

ESAC,

out= MXT{STRING[scratch_exp-6]bit]{b*0111111111111,b*10000000000,sum[7..scratch_exp],sum[7..scratch_exp],soa,t,ss1}. OUTPUT (S_TO_IN out)[2]

END.

MAC LINE_DELAY_ST(TYPE t)=([4]t:in,t_col:wr_address,t_col:rd_address,t_load.rw)

至

RAM([4]?1).

FN PR_ADDER_ST = (L_scratch.a b)

(S_TO_!(scratch_exp) ADD_S((I_TO_S(scratch_exp-1)a)[2],(I_TO_S(scratch_exp-1)b)[2])) [2]. t_scratch:

FN ADD_SUB_ST = (t_scratch: a b, t_add:sel)

t_scratch: BEGIN LET a_s=(I_TO_S(scratch_exp)a)[2],
b_s=(I_TO_S(scratch_exp)b)[2],
sel_bit = CAST(STRINQ[1]bit]sel,

#ACTEL# b_s_inv = XOR_B[scratch_exp](b_s, ALL_SAME[scratch_exp]sel_bit),

#cinb is active tow so cast sel(add->0,sub->1) & invert it#

out= ADD_S_ACTEL(a_s, b_s_inv,CAST(bit)INV(1)sel_bit),
binout= out[2..scratch_exp+1].

OUTPUT (S_TO_I[scratch_exp]binout)[2] END.

MAC ALL_SAME(INT n) = (STRING(1)bit:dummy)

STRING[n]bit:

BEGIN

FAULT IF n < 1 THEN "N<1 in ALL_SAME" FI.
OUTPUT IF n=1 THEN dummy

ELSE dummy CONC ALL_SAME[n-1] dummy

END.

MAC CAST (TYPE to) = (TYPE from:in)

to:ALIEN CAST.

```
MAC ZERO(INT n) = (STRING[1]xit:dummy)
```

STRING[n]bit:

BEGIN

FAULT IF n < 1 THEN "N<1 in ZERO" FI.

ELSE b'0" CONC ZERO(n-1) b'0" OUTPUT IF n=1 THEN b-0"

MAC B_TO_S= (bit:in)

STRING[1]bit: CASE in

OF **b**0:**b***0.

MAC I_TO_S(INT n) = (I_scratch: in)

(flag,STRING[n]bit): BIOP TRANSFORM_S. MAC S_TO_[(INT n) = (STRING[n]bit:in)

(flag,t_scratch): BIOP TRANSFORM_S. MAC S_TO_IN = (STRING[input_exp]bit:in)

(flag,t_input): BIOP TRANSFORM_S. MAC IN_TO_S(INT n) = (t_input: in)

(flag,STRING[n]bit): BIOP TRANSFORM_S. MAC U_TO_I(INT n) = (STRING[n]bit:in)

```
(flag,t_scratch): BIOP TRANSFORM_U.
```

MAC B_TO_l= (bit:in)

-> t_scratch: CASE in

ESAC. MAC CARRY= (L_add:in)

OF b'0:scratch/0, b'1:scratch/1

OF add:b*0., sub:b*1*

-> STRING(1]bit: CASE in

ESAC.

MAC BOOL_BIT = (bool:in)

STRING[1] bit: CASE in OF t.b.1"

ESAC. MAC BIT_BOOL= (bit:in) ELSE b'0"

```
MAC BOOL_STRING(INT n) = ([n]bool:in)
```

STRING[n] bit: (LET out = BOOL_BIT In[1]. OUTPUT IF n=1

THEN out ELSE out[1] CONG BOOL_STRING[n-1](in[2..n]) FI

FN NOT = (bool:in) ->bool: CASE in OF t:f, #define a few useful gates #

ESAC. FN MUX = (bool:set int in2) -> bool:

two input mux, select in1 if sel =t,otherwise in2 # CASE sel

tin2, tin1

FN XNOR=(boolin1 In2)->bool: CASE (in1,in2) OF (f.f)1, ESAC.

ESAC.

1bit input to binary # ->**bool**: FN INT_BOOL=(1_input:k)
CASE k

input/0:f, input/1:t

ESAC.

1 bit bool to input # FN BOOL_INT=(bool:b) ->t_input:

finput/0, CASE b OF ting

t:input/1

ESAC.

->1_input: ARITH a*b. FN * =(t_input:a b)

A_input: ARITH a%b. FN % =(t_input:a b)

'> input: ARITH a+b. >1_input: ARITH a-b. FN - =(t_input:a b) FN + =(t_input:a b)

> Lest: ARITH IF a=b THEN 2 ELSE 1 FI. FN = =(t_input:a b)

#changes sign for 8-bit 2's# #complement no, # FN CHANGE_SIGN = (!_input:) ->t_input: ARITH IF I<0 THEN 128+i #

ELSE i F1.

FN SIGN = (t_input:i) ->bool: ARITH IF I<0 THEN 1

#gets sign for 2's# #complement nos #

ELSE 2 FI.

FN TEST_SIZE = (1_inputx)

```
#checks to see if orig will#
                                                                                                                                                                                                                                                                                                                                                                                                                               #converts 8bit boolean to 2's#
                                                                                                                                                                                                                                                                                                                                    #fit inputo an 8_bit value#
                                                                                                                                                                                                                                                                                                                                                                                                                                                                  #complement inputager #
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        eum:≍eum+exp*BOOL_lNT(bfk]);
                                      ビ
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   VAR sum = input/-128 * BOOL_INT(b/8))
#tests to see if the Irput is bigger than an 8-bit inputeger#
ARITH IF ( (x<=-128) AND (x>127)) THEN 1
                                                                                                                                                                                                                                                           bin]:=INT_BOOL(io-input/2*i1)
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       exp:=input/2 * exp
                                     ELSE 2
                                                                                                                                                                                                                                                                                                               CASE TEST_SIZE orig
                                                                                                                                                                 VAR it:=input/0, #input 10:=CHANGE_SIGN(orig), b:=(I,I,I,I,I,I,SIGN(orig));
                                                                                                                                                                                                                                                                                                                                                                                                                               -X input:
                                                                                           FN INT8_BOOL=(t_input:orig) ->[8]bool:
                                                                                                                                                                                                                                                                                                                                     [8]?bool,
b
                                                                                                                                                                                                                                           1:=i0%input/2;
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     exp:=input/1;
[INT k=1..7]
                                                                                                                                                                                                                                                                               <u>==:0</u>
                                                                                                                                                                                                                                                                                                                                                                                                                              FN BOOL_INT8=([8]bool:b)
                                                                                                                                                                                                                                                                                                                                                                        ESAC
                                                                                                                                                                                                                         [INT n=1.7]
                                                                                                                                                                                                                                                                                                                                     9
                                                                                                                                                                                                                                                                                                                 OUTPUT
                                                                                                                                                                                                                                                                                                                                                                                                                                                                SEQ
                                                                                                                                                 SEO
                                                                                                            BEGIN
                                                                                                                                                                                                                                                                                                                                                                                          ENO.
```

OUTPUT sum

END.

```
#hack because of sign extend#
                                                                                                                                                                                                                                                                                                                                                (BOOL_INTB(in1))+((input/256)*BOOL_INTB(in2))+((irpu1/256)*BOOL_INT(in1[8])
        #converts 10bit boolean to 2's#
                                                                                                                                                                                                                                                                                                                                                                                                       #of lab #
                                                                                                                                                                                                                                                                                                                                                                                                                                                #compute the mean equare difference between two arrays of integers#
                                                                                                                                                    sum:=sum+exp*BOOL_INT(b[k]);
                                                  #complement integer
                                                                          8um input/-512 * BOOL_INT(b[10]),
                                                                                                                                                                                                                                                                                                                         # convetrs a 16-bit no., (labs.mabs) inputo inputeger form)#
                                                                                                                                                                              exp:=input/2 exp
                                                                                                                                                                                                                                                                                                        ->t_input:
FN BOOL_INT10=((10)bool:b) ->t_input:
                                                                                                      exp:=input/1;
                                                                                                                                                                                                                                                                                                FN BOOL_INT16 =([8]bool:in1 In2)
                                                                                                                            [INT k=1.9]
                                                                                                                                                                                                                            OUTPUT sum
                                                                          VAR
                                                 SEO
                                                                                                                                                                                                                                                                                                                                                                                                                        ₩
```

FN MSE_COLOUR = (L_reset:reset,t_input:a b) ->[2]t_int32:

PDEL(t_reset,0):edge.

rising = CASE (reset, edge) ᄪ

(no_rat,rat):diff32, (no_rat,no_rat):del PL diff32

ELSE ESAC.

żą. reset JOIN rising

OUTPUT END.

MAKE SAVE_ERROR:save_error. LET out = (SEQ

STATE VAR true_count INIT int32/1 VAR diff:=int32/0,

diff32:=Int32/0,

incr:=int32/0;

diff:=CASE reset

ELSE I_32(a) MI I_ ESAC; OF retrint32/0

OF retaht32/0 incr:=CASE reset

ELSE Int32/1

ESAC;

true_count:= CASE reset

OF ret int32/1 ELSE true_count PL_incr ESAC;

diff32:= (diff TI diff);

OUTPUT (diff32,true_count)).

(reset,out[1]) OUTPUT END.

(save_error,save_error DV out[2])

#compute the mean equare difference between two arrays of integers#

TYPE 1_int32 = NEW Int32/(-2147483000..2147483000). INT period_row=9.

->t_int32:ARITH in. \$_32 = (t_input:in) ->t_int32:ARITH a%b. ->t int32:ARITH a+b. FN PL = (L_int32:a b) FN MI = (L_int32:a b) FN DV = (t_int32:a b)

-> int32:ARITH a-b. ->1 Int32:ARITH a.b. FN TI = (L_int32:a b) FN MSE_ROW = (L_input:a b) ->[3]t_int32: BEGIN SEQ

STATE VAR en INIT int32/0, count INIT int32/0;

VAR diff:-int32/0, diff32:=Int32/0;

count:=count PL int32/1;

OF int32/(1..period_row).int32/0 ELSE I_32(a) MI I_32(b) ESAC; diff:=CASE count

diff32:= (diff TI diff); err:=err PL diff32;

OUTPUT (err, err DV count, count)

#A 10 bit prbs generator, feedback taps on regs 3 & 10.# BEGIN -×10bool: FN PRBS10 = (t_reset:reset)

MAKE [10]MYLATCH: XNOR:xnor.

FOR INT k=1..9 JOIN (reset,[[k])

->[1], ->mor. (reset,xnor) ([10],[3]) NOS

OUTPUT

FN PRBS11 = (L_reset:reset)

#A 11 bit prbs generator, feedback taps on regs 2 & 11.#

MAKE [11]MYLATCH1, XNORxnor.

FOR INT k=1..10

->[k+1]. (reset,[[k])

->(T), ->xnor. (reset,xnor) (l[11].[2]) NOC

[1..10] OUTPUT

FN PRBS16 = (bool:reset)

#A 16 bil prbs generator,feedback taps on regs 1,3,12,16# BEGIN ->(16]bool:

MAKE [16]MYLATCH1, XOR_4:xor,

NOT:xnor.

(ck,reset,f[k]) FOR INT k=1..15

N S

->[k+1].

(ck,reset,xnor) ->[[1], NOS

(((1),(3),(16),(12)) xor

(|INT k=1..16][k]) OUTPUT

#A 12 bit pribs generator, feedback taps on regs 1,4,6,12.# ->[12]bool: FN PRBS12 - (clock:ck,bool:reset) BEGIN

MAKE [12]MYLATCH:1,

XOR_4:xor, NOT:xnor.

FOR INT k=1..11 JOHN (ck,reset,[k]) ->[k+1].

JOIN (ck,reset,xnor) ->[1],
([[1],[4],[6],[12]) ->xor,
xor

OUTPUT ((INT k=1..12)(K))

ES

FN PRBS8 = (clock:ck,bool:reset) ->{8]bool: #A 8 bit prbs generator,feedback taps on regs 2,3,4,8.# BEGIN

MAKE [8]MYLATCH:1, XOR_4:xor, NOT:xnor. FOR INT k=1..7 JOIN (ck,reset,f[k]) ->f JOIN (ck,reset,xnor) ->[1], ([[2],[3],[4],[8]) ->xor,

KOT ->x00C. ->x0OT ->X0OC. [INT k=1..8][K]

OUTPUT

ENO.

#TEST FOR Y U V#

#to test the 2d convolver using pros input into the forward convolver# #then outputting to the inverse convolver and checking against the original result# FN TEST_COLOUR = (bool:ck,t_reset:reset,bool:extwritel_in csl_in, t_sparc_addr:reg_set value,t_reset;prbs_reset) ->[3]t_int32:

BEGIN

FN DEL = (I_load:in) ->I_load:DELAY(read,1).

FN PULSE = (Lloadin) -> Lreset: CASE (in, DEL in)

(write, read):rst

5 D ELSE

ESAC.

MAKE PRBS11:prbs,

BOOL_INT10:int_bool DWT:dwt,

[3]MSE_COLOUR:mse_colour.

(CASE (prbs_reset, PULSE CASE dwt[3][2] NO S

OF write:read,

ESAC, PULSE CASE AM(3)[3] read:write

OF write read,

ESAC, PULSE AMIZII 1], PULSE AMIZIIZI, PULSE AMIZIIZI)

(rst,t_reset,t_reset,t_reset,t_reset)((t_reset,ret,t_reset,t_reset,t_reset,t_reset,t_reset)|

P

_reset,t_reset,reset,t_reset,t_reset)||(_reset,t_reset,t_reset,reset,reset,t_r

ESAC)

Frenun the pros at start, or on out of IDWT#

(ck,reset,int_bool,extwritel_in,cal_in, reg_set,value) ->int_bool

-×det

#calcuate the mse error for each channel#

FOR INT J=1.: B JOIN (CASE AMIEILI)

OF read:nst

ELSE no_rat

ESAC, dwt[1], int_bool) -> msa_colour[]]

OUTPUT (mse_colouif1][1].mse_colouif3][1]].mse_colouif3][1]]

FN DWT = (boof,t_reset,t_input,boof,boof,t_sparc_addr:reg_set value) MAC PDEL(TYPE t, INT n) =(t) ->t:IMPORT.

->(t_input,[3]t_load,[3]t_load):IMPORT.

dwt/string: DWT_TEST(RENAMED DWT) PDEL

IMPORTS

#TEST FOR LUMINANCE ONLY#

#then outputting to the inverse convolver and checking against the original result# #to test the 2d convolver using price input into the forward convolver#

FN TEST_Y = (bool:ck,t_reset:reset,bool:extwritlel_in csl_in, t_sparc_addr:reg_set value,t_reset.prbs_reset)

BEGIN

->(_load:DELAY(read,1). FN DEL = (I_bad:in)

FN PULSE = (t_load:in) ->t_reset: CASE (in,DEL in)

(write,read):rst MAKE PRBS11 prbs 19 19 19 OF ELSE ESAC.

BOOL_INT10.int_bool, DWT:dwt, MSE_COLOUR:mse_colour.

#rerun the pros at start, or on out of IDWT# (CASE (prbs_reset,PULSE dwt[2][1])
OF (rst,1_reset)]((_reset,rst):rst N_O

ELSE ESAC)

->prbs.

prbs ->int_bool, (ck,reset,int_bool,extwritel_in,cal_in, reg_sel,value)

(CASE dw[2][1]

OF read:rst

ELSE no_rst ESAC,dwr[1],int_bool) OUTPUT mse_colour

ENO.

APPENDIX B-2

#test for abs #

FN ABS_TEST = (STRING(10)bit:in in2) ->bool: in LE_U in2. flonly works for 3 octave decomposition in y/2 in ulv# #the state machine to control the address counters#

FN CONTROL_ENABLE = (bool:ck,t_reset:reset,t_channel:new_channel.channel,[3]boot:c_bfk,STRING[2]bit:subband, t_load:load_channel, t_mode:new_mode)

->([3]bool#en_bik#,t_octave,[2]bool#iree_done,tpf_block_done#,t_state#reset_state#):

MAKE DF1(t state):state.

#set up initial state thro mux on reset, on HH stay in zzo state# LET start state = CASE channel

OFulv:down1,

y:up0

reset_state= CASE reset ESAC,

OF rst: start_state ELSE state

ESAC.

VAR en_blic=[3]f, #enable blk_count# LET next_values = (SEQ

#enable x_count for other subbands# #enable x count for LPF# tot block done:=f,

tree_done:=f, #enabk new_state:=reset_state,

oclave:=?t_oclave; #current oclave#

CASE reset_state

en_bik[1]:=t; CASE c_bik[1] OF t:(new_state:=zz1;

zz0: (octave:=oct/0;

ESAC),

ESAC)

```
OF upo: ( octave:=oct/2;

en_blk[3]:=t;
CASE c_blk[3]
OF t:(CASE subband
OF b*00*:pf_block_done:=t #clock x_count for LPF y channel#
ELSE new_state:=up1 #change state when count done#
ESAC;
CASE new_mode #in luminance & done with that tree#
OF stop:tree_done:=t
ELSE
ESAC)
ELSE
ESAC)
Up1: ( octave:=oct/1;
en_blk[2]:=t;
CASE new_mode #in luminance, terminate branch & move to next branch#
OF t:(new_state:=down1;
en_blk[3]:=t)
ELSE
EASE
CASE new_mode #in luminance, terminate branch & move to next branch#
OFstop:(new_state:=down1;
en_blk[3]:=t)
ELSE
```

```
#stop so finish this tree/branch & move on#
         #dock x_count for LPF ulv channe#
             OF b"00":Ipf_block_done:=t #dock x_count for LPF ulv ELSE new_state:=zz0 #change state when count done#
                                                                                                                                   imove to next treef
                                                                                                                                                                     ELSE new_state:=down1
ESAC
                                                                                                                                                                                                                                                                                                                                                                                                                             y: CASE (c_bk[1],c_bk[2],c_bk[3])
OF (t,t,f):tree_done:=t
ELSE
ESAC
                                                                                  CASE (new_mode,channel)
OF (stop,u|v):tree_done:=t,
                                                                                                                (stop,y):(en_blk[3]:=t;
CASE c_blk[3] |
OFt:tree_done:=t
                                                                                                                                                                                                                                                                                                                                                           OF up: CASE (c_blk(1],c_bk(2))
OF (t,t).tree_donec=t
t:(CASE subband
                                                                                                                                                                                                                                    ESAC)
ELSE
ESAC)
                                                                                                                                                                                                                                                                                                                                                                                                              ESAC,
                                                                                                                                                                                                                                                                                                                                                                                            ELSE
                                                   ESAC;
                                                                                                                                                                                                                                                                                                                                             CASE channel
 P
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   ESAC;
                                                                                                                                                                                                                                                                                                            ESAC;
```

tree done #in LPF state doesnt change when block done#

OFt: new state:= start state

ELSE

CASE

#now change to start state if the sequence has finished#

ESAC;
#on channel change, use starting state for new channel#
CASE load_channel #in LPF state doesnt change when block done#
OF write: new_state:= CASE new_channel
OFy:up0,
u|v:down1
ESAC
ELSE

OUTPUT (new_state,en_bik,octave,(tree_done,ipf_block_done))
).

JOIN (ck.next_values[1]) ->state. OUTPUT (next_values[2],next_values[3],next_values[4],reset_state) END. FN CHECK = (t_inputx sub size y,t_octave: oct) ->t_sparc_addr: ARITH ((x SL 1)+(1 IAND sub) + size*((y SL 1) +(sub SR 1)))SL oct. #these are the addr gens for the x & y adresses of a pixel given the octave# #sub&blk no. for each octave.Each x & y address is of the form # # x= count(5 bits) (bik(3)..bik(octave+1) {s} {octave 0's} #

#by write_enable, so same address values generated on read & write cycles# #carry-outs for the mode change, this is done on the write addr cycle # fread enable enable the block count for the read address, but not the #the bik & s counters are vertical 2 bit with the lsb in the x coord # and carry out on 3, last counter is both hortz and vertical counter# #this makes up the 9 bit address for CIF images

v= count(5 bits){blk(3)..blk(octave+1)}{s} {octave 0's}

FN ADDR_GEN = (bool:ck, t_reset:reset,t_channel:new_channel channel,t_load:load_channel,STRING[2]bit:sub_count, STRING[xsize]bit:col_length,STRING[ysize]bit:row_length,STRING[xsize]bit:col_length,STRING[ysize]bit:row_length,STRING[xsize]bit:col_length,STRING[ysize]bit:row_length,STRING[xsize]bit:col_length,STRING[ysize]bit:row_length,STRING[xsize]bit:col_length,STRING[ysize]bit:row_length,STRING[xsize]bit:col_length,STRING[ysize]bit:row_length,STRING[xsize]bit:col_length,STRING[ysize]bit:row_length,STRING[xsize]bit:col_length,STRING[ysize]bit:row_length,STRING[xsize]bit:col_length,STRING[ysize]bit:row_length,STRING[xsize]bit:col_length,STRING[ysize]bit:row_length,STRING[xsize]bit:col_length,STRING[ysize]bit:row_length,STRING[xsize]bit:row_length,ST STRINGlysizejbit.yimage_string,STRING(11]bit.yimage_string_3#yimage^2.5# bool:read_enable write_enable, t_mode:new_mode)

-> (t_sparc_addr,t_octave,bool#sub finished#,bool#tree_done#,bool#tof_done#,t_state);

CONTROL_ENABLE:control, [3]BLK_SUB_COUNT:blk_count. MAKE COUNTER(xsize-4):x_count, COUNTER{ysize-4}:y_count,

#ipf same size for all channels!!!#

size of lpf images/2-1, for y, u/v. /2 because count in pairs of lpf values #

LET (x_lpf,y_lpf) = (col_length[1..xsize-4], row_length[1..ysize-4])

x_en = CASE (tree_done,lpf_block_done) tof block done = control[3][2], tree_done = control[3][1], OF(t,bool)|(bool,t);t ELSE 1 ESAC, #cik y_count when all blocks done for subs 1-3, or when final bik done for toff#

bik_en=control[1], octave=control[2], y_en = CASE sub_count OFb*00*:CASE ([pf_block_done, x_count[2]) OF(t,t):t

OF(I,I):1 ELSE f

ESAC

ELSE CASE (tree_done, x_count[2])

OF(t,t),t ELSE f

ESAC

ESAC.

x_msb_out = CASE channel

#always the msb bits#

OF y: x count(1) CONC B TO_S(blk_count(3)[1)[2]), ulv: b*0* CONC x_count(1)

OF y.y count(1) CONC B TO S(blk_count(3)[1][1]), ulv.b.0* CONC y_count(1)

y_msb_out = CASE channel

x_lsb_out =CASE (octave) #bit2 ls lsb#
OF(oct/0):([INT k=1..2]bik_count[3-k][1][2])CONC sub_count[2],
(oct/1):(bik_count[2][1][2], sub_count[2], b'0),
(oct/2):sub_count[2] CONC [2]b'0

ESAC.

(oct/0):([INT k=1.2]blk_count[3-k][1][1][1]CONC sub_count[1], (oct/1):(blk_count[2][1][1], sub_count[1], b'0), (oct/2):sub_count[1] CONC [2]b'0 x_addr = x_msb_out CONC BIT_STRING(3)x_lsb_out, y_addr = y_msb_out CONC BIT_STRING(3)y_lsb_out, sub_en = CASE (y_count[2],y_en)
OF (t,t),t #enable the sub band counter# y_lsb_out =CASE (octave) ESAC, ESAC,

#bit 1 is msb#

#IIIICHANGE ACCORDING TO LATENCY IN DECODE# tpf_done = CASE sub_count OF b'00": sub_en ELSET ESAC,

base y_sel = CASE channel ESAC, 7

base_rows = MUX_3{STRING[11]bit]{ZERO(11]b'0',b'0' CONC yimage_string[1..ystze]CONC.b'0', ymage_string_3,base_y_sell, #base address for no of rows for y,u &v memory areas#

address = x_addr ADD_U ((y_addr ADD_U base_rows)[2..12]) MULT_U (CASE channel OFyzdmage_string,

ulv:(SR_U(1)ximage_string)[1.xslze] ESAC)

int_addr = (S_TO_SPARC address)[2]

->x_count, (ck,reset,x_en,x_lpf) (ck,reset,y_en,y_lpf)

->y count,

(ck,reset,new_channel,channel,([INT j=1..3]blk_count[[[2]],sub_count,load_channel,new_mode) #use new_channel so on channel change control state picks up correct value

->control.

->bik_count[k]. FOR INT k=1.3 JOIN (ck,reset,blk_en[k],read_enable OR write_enable)

OUTPUT (int_addr,octave, sub_en,tree_done, ipt_done,control[4])

#ipf stop is a is a dummy mode to disable the block writes&huffman data# #decide reset is enabled 1 cycle early, and latched to avoid glitches# a counter to control the sequencing of r/w, token, huffman cycles# tcycles for that block# FN CONTROL_COUNTER = (bootick,t_resetreset,t_mode:mode.new_mode,t_direction.dfrection) ->(i_load,t_cycle,t_reset,bool,bool,t_load,t_cs,t_load,t_cs):

#decode write_addr_enable early and latch to avoid feedback loop with pro_mode# #mode load,cycle,decide reset,read_addr_enable,write_addr_enable,load flags#

Hin MODE CONTROLA

MAKE COUNT_SYNC(4):count

LET count_len = (U_TO_LEN{4} count[1])[2].

LET out = (SEQ

VAR cycle:=skip_cycle,
decide_reset:= no_rst,
load_mode:=read,

toad_flags:=read, cs_new:=no_select, cs_old:=select, rw_old:=read, read_addr_enable:=f, write_addr_enable:=f, CASE direction
OFforward: CASE mode
OF send|still_send|lpf_send: CASE count_len
OF send|still_send|lpf_send: CASE count_len
OF tert/(0..3):(read_addr_enable:=t;
cs new:=select),

len/(4):(cycle:=token_cycle; load_flags:=write; write_addr_enable:=t),

stoplipf_stop:(cyde:=skip_cyde; rw_okd:=read; cs_old:=no_select), len/8:(decide_reset:=rst; CASE new mode

load_mode:=write;

ELSE (cycle:=data_cycle; load mode:=write;

rw_old=write)

ELSE ESAC,

ij

count len | len/lo.3):(read_addr_enable:=t; CASE OF

len/(5..7):(rw_old:=write;

wite_addr_enable:=t; CASE new_mode OF void still:cycle:=skip_cycle

ELSE cyde:=data_cycle

len/8:(decide_reset:=rst;

CASE new mode OF void still:cycle:≔skip rw_old:=write; load mode:=write; ELSE ESAC)

ELSE ESAC,

CASE count len
OF len/(0..3):(read_addr_enable:=t;

lan/(4):(cycle:=token_cycle; cs new:=select),

write_addr_enable:=t load flags:=write),

len/(5..7):(cyde:=data_cycle;

rw_old:=write; write_addr_enable:=t) len/8:(cycle:=data_cycle;

decide_reset:=rst; rw_old:=write;

ELSE ESAC,

CASE count_len OF len(0.3);(read_addr_enable:=t;

cs_new:=select),

len/4:(load_flags:=write; cycler=token_cycle; #dummy token cycle for mode update#

write_addr_enable:=t), len/(5..7):(write_addr_enable:=t; #keep counters going# stop:(m_old:=read; cs_old:=no_select) stop:(rw_old:=read; cs_old:=no_select) ESAC), len/8:(decide_reset:=rst; ELSE (load_mode:=write; rw_old:=write) ELSE rw_old:=write P CASE new_mode CASE new mode OF ESAC)

ELSE ESAC,

OF Ten/0: write_addr_enable:=t, #allow for delay# len/(1..3):(write_addr_enable:=t; rw_old:=write), CASE count_len void_stilt

decide_reset:=rst) ELSE ESAC

len/4:(rw_old:=write; load_mode:=write;

ELSE ESAC,

inverse: CASE mode

ELSE (cycle:=data_cycle; load_mode:=write; rw_old:=write)

ELSE ESAC,

```
#skip to allow reset in huffman#
                                      len/(1):(cycle:=token_cycle;
write_addr_enable:=1),
len/(2..4):(rw_old:=write;
CASE count len
OF lerv(0):,
   葛
```

write_addr_enable:=t;

CASE new_mode_OFvoid_stillcycle:=skip_cycle ELSE cycle:=data_cycle

ESAC),

decide reset:=rst; len/5:(rw_old:=write;

load mode:=write; CASE new mode

OF void_still:cycle:≂skip_cycle ELSE cycle:=data_cycle

ESAC)

#match with previous #skip for write enb delay ELSE ESAC, CASE count len OF len/(0);

write_addr_enable:=1), len/(2..4):(cycle:=data_cycle;

rw_old:=vnite; write_addr_enable:=t),

len/5:(cydec=data_cyde;

oad_mode:=write) decide_reset:=rst; rw old:=write;

```
cycle:=token_cycle; #dummy token cycle for mode update#
                                         OF len/(0..3):(read addr_enable:=t),
                                                                                                                                                                           stop:(rw_old:=read
                                                                                                                                len/(5..7):(write_addr_enable:=t;
                                                                                                           write_addr_enable:=t)
                                                                len/4: (load_flags:=write;
                     CASE count len
                                                                                                                                                   CASE new_mode
ESAC,
                       void:
```

len/8:(decide_reset:=rst; ESAC),

ELSE rw_old:=write

cs old:=no select CASE new mode OF stop:(rw_old:=read

ELSE (load_mode:=write; rw_old:=write)

ESAC)

ESAC,

CASE count_len

| len/1:write_addr_enable:=t, #dummy as write delayed# | len/(2..4):(write_addr_enable:=t; #match with rest# OF lan/(0):,

len/5: (rw old:=write

decide_reset:≂rst) ELSE load mode:=write;

ESAC

ELSE ESAC

SAC;

OUTPUT (load_mode,cycle,DF1{t_reset}{ck,decide_reset},read_addr_enable, DFF{boot}{ck,reset,write_addr_enable,f},load_flags, cs_new,rw_old,cs_old)

JOIN (ck,CASE reset
OF retriet
ELSE out[3]
ESAC,() ->count.

OUTPUT out END.

#A set of boolean ,ie gate level counters # #The basic toggle flip-flop plus and gate for a synchronous counter #input t is the toggle ,outputs are q and tc (toggle for next counter# stage #

MAC BASIC_COUNT = (bool:ck, i_reset:reset;bool: tog) ->(STRING[i]bit,bool):

```
MAC COUNT_SYNC(INT n) = (bootck,t_reset: reset,boot: en )->(STRING(n]bit,boot): (LET out = BASIC_COUNT(ck,reset,en).
                                                                                                                                                                                                                                                                                                                                            #are msb(bit 1).....ksb,carry.This is the same order as ELLA strings are stored#
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                FN TEST_COUNT_SYNC = (bool:ck,t_reset: reset,bool: en ) ->[[4]bool,bool):
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        ELSE (LET outn = COUNT_SYNC[n-1](ck,reset,out[2]).
OUTPUT (outn[1] CONC out[1],outn[2])
                                                                                                                                                                                                                                                                                                               # The n-bit macro counter generator, en is the enable, the outputs #
                                                                                                                                                                      (tog,dlat) ->xor.
OUTPUT (CAST(STRING[1]bit) dlat,and)
                                                                                                                                                                                                                                                                                                                                                                                COUNT_SYNC(4) (dk,reset,en)
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              THEN (out(1),out(2))
                                                                                                                JOIN (ck,reset,xor,f)->dlat,
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      Fn=1
MAKE DFF{bool}:dlat,
                                                                                                                                            (diat,tog) ->and,
                             XOR xor,
AND xand.
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               E
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      OUTPUT
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         ₩
```

tc (toggle for next counterstage, active low for down/high for up) #

#input t is the toggle, updown detms the direction ,outputs are q and # #The basic toggle flip-flop plus and gate for a synchronous counter #

MAC BASIC_COUNT_UD = (bool:ck, t_reset:reset,bool: tog, t_updown:updown) ->[2]bool:

BEGIN

MAKE DFF(boof):dlat. toggle = tog, 回

xorn = CASE updown

OF up: CASE (toggle,dlat) #xor# OF (t,t)(f,f):f

ELSEI

down:CASE (toggle,dlat) #xnor# OF (t,t)[(f,f):t ELSE f

ESAC

ESAC,

OFup:CASE (diat,toggle) #AND# cout = CASE updown

OF (t,t):t ELSE f

ESAC,

down:CASE (dlat,toggle) #OR#

OF(f,f):f ELSE t ESAC

ESAC.

JOIN (ck,reset,xorn,f)->dlat. OUTPUT (diat,cout)

/are msb(bit 1).....lsb,carry.This is the same order as ELLA strings are stored# IThe n-bit macro u/d counter generator, en is the enable, the outputs ffirst enable is active low on down, so invert.

MAC COUNT_SYNC_UD{INT n} = (bool:ck,t_reset: reset,bool:en, t_updown:updown) ->(STRING[n]bit,bool):

MAKE [n]BASIC_COUNT_UD:basic_count. LET enable = ([INT k=1..n-1] basic_count[k+1][2]) CONC CASE updown

#invert enable if down count#

ELSE NOT en OF up:en

ESAC.

OUTPUT (BOOL_STRING(n) (IINT k=1..n]basic_count[k][1]], basic_count[1][2] FOR INT k=1..n JOIN (ck,reset,enable[k].updown) ->basic_count[k]

8

FN TEST_COUNT_SYNC_UD = (bod:ck,t_reset: reset.boot: en,t_updown:updown:) ->[[4]bool,bool): COUNT_SYNC_UD[4](ck,reset,en,updown).

20€

#the basic xy counter, carry out 1 cycle before final count given by $x_{\perp}[pt]y_{\perp}[pt]$ =>(STRING[n]bit,bool): MAC COUNTER[INT n] = (bool:ck,t_reset.reset,bool:en,STRING[n]bit.x_[tpf] ->(STRING[n]bit,bool):

```
MAKE COUNT_SYNC(n):x_count.
```

LET out = x_count[1],
final_count = out EQ_Ux_[pt,
final_count_en=CASE (final_count,en)
OF_(t,t):1
ELSE f
ESAC,
#reset after 4 counts at final count value#
cnt_reset = CASE reset

OF rst:rst ELSE CASE DF1{bool}(ck,final_count_en) #reset taken out of DFF 12/6#

OFfirst

ELSE no_rst

ESAC ESAC.

JOIN (ck,cnt_reset,en) ->x_count.
OUTPUT (out,final_count)

END.

COM.

MAC Y_COUNTER = (bookch, resetreset, booken, STRING[4]bit, y_lpf) ->(STRING[4]bit, book):
BEGIN ithe basicy counter, carry out 1 cycle before final count given by y_tpf#

MAKE COUNT_SYNC{4}:y_count.

LET out = y_count(1). JOIN (ck,reset,en) ->y_count. OUTPUT (out, out EQ_U y_lpt)

MOC

#the bik, or sub-band counters, carry out on 3#

->(STRING[2]bit,bool): FN BLK_SUB_COUNT = (boot:ck,i_resetreset, boot:en)

MAKE COUNT_SYNC[2]:bik_count.
LET out = bik_count[1].
JOIN (ck,resel,en) ->bik_count.
OUTPUT(out,out EQ_U (C_TO_S(2)col3)[2])

MOC

#the bit, or sub-band counters, carry out on 3, cout_en enables the carry out, & dn_en AND en enables the count# ->(STRING[Z]bit,bool):

FN BLK_SUB_COUNT = (bookd, t_resetzeset, booken dn_en cout_en)
BEGIN

MAKE COUNT_SYNC(2):bik_count.

LET out = bik_count[1].

JOIN (ck,reset,en AND cin_en) ->bik_count.

OUTPUT(out,(out EQ_U (C_TO_S(2)cot(3)[2]) AND cout_en)

FN LAST_BLK_COUNT = (bool:ck1_reset:reset, bool:en,1_channel:channel,boot:line_finished) ->

(STRING[Z]bit, [Z]bool#x_en,y_en#):

MAKE BASIC_COUNT: Isb msb.

JOIN (ck,reset,en) ->ksb,

(ck, reset, CASE channel

OF y:sb[2],
u(v:line_finished
ESAC) ->msb.
LET out = (msb[1]CONClsb[1]).
OUTPUT (out, CASE channel
OF y:(out EQ_U (C_TO_S[2]col/3)[2],line_finished),
u(v:(sb[2],msb[2])
ESAC)
END.
#the L1 norm calculator/ comparison constants& flag values#
#adding 4 absolute data values so result can grow by 2 bits#
#5 cycle sequence, a reset cycle with no data input, followed#

MAC L1NORM = (bool:ck, t_reset:reset, STRING[INT n]bit:in) ->STRING[n+2]bit: BEGIN

#by 4 data cycles#

MAKE DF1{STRING[n+4]bit}:in2.

LET in s =in,

msb = ALL_SAME{n}{B_TO_Sin_s{1}},

COM

add_in1 = in2 CONC in_s{1}, #in_s{1} is the carryin to the ædder#

add_in2 = ((in_s XOR_B msb)CONC in_s{1}),

#adder=ADD_U(add_in1,add_in2),#

MOC

add_in1 = (in_s XOR_B msb),

rst_mux = CASE reset

OF rst:ZERO(n+4)b'0*

ESAC,

adder=ADD_US_ACTEL(add_in1,rst_mux,CASE in_s[1] OF b1:b10

ELSE 6'1

ESAC),

out =adder[2..(n+5)].

JOIN (ck,out) ->in2.

OUTPUT in2[3..n+4]

FN ALL_ZERO = (bool:ck, t_reset:reset, t_input:in) ->bool: BEGIN #the block to decide if all its inputs are all O#

MAKE DF1{bool}:out.

LET in_s = (IN_TO_S(input_exp)in)[2],

in_eq_0 = in_s EQ_U ZERO(input_exp)b*0*, #in =0# #1 it reset high, & OR with previous flag#

all_eq_0 = CASE reset
OF_rst: In_eq_0
ELSE CASE out

ELSE in_eq_0 ESAC ESAC.

```
MAC ABS_NORM = (bool:ck, t_reset:reset,STRING[result_exp-2]bit:qshift, STRING[INT njbit:in) ->(STRING[n+2]bit,boolfalf <qshiftf):
```

JOIN (ck,all_eq_0)->out. OUTPUT out

BEGIN

MAKE DF1{STRING[n+4]bit}:in2,

DF1{bool}:out.

T abs_in = ABS_S in, rst_mux = CASE reset

ELSE in2

ESAC,

adder = ADD_US_ACTEL(abs_in,rst_mux,b*1), add_s =adder[2..(n+5)],

in small = abs in LT U qshift, #1 if reset high, & OR with previous flags all small = CASE reset

ELSE CASE in small OF 1st: t

ELSE out ESAC

ESAC.

JOIN (ck,add_s) ->in2, (ck,all_smail) ->out.

OUTPUT (in2[3..n+4],out)

FN DECIDE = (boot:ck,t_reset:reset,t_result:q_int,t_input:new old, t_result: threshold comparison, ->77 bool: t octave:octs,t load:load flags) #the decide fn block#

#nzflag,origin,noflag,ozflag,motion.pro_new_z.pro_no_z#

MAKELINORM (Input_exp): 02,

ABS_NORM(input_exp): nz, ABS_NORM(input_exp+1):no,

LATCH([7]bool]:flags.

LET qshift=(I_TO_SC{result_exp}q_int)[2][1..result_exp-2]. #divide by 4 as test is on coeff values not block values#

n o =(IN TO S(input_exp)new)[2] SUB_S (IN TO S(input_exp)old)[2], #new-old,use from quant# nzflag = rz[1] LE_U (i_TO_SC(result_exp)threshold)[2], #delay tests for pipelined data# noflag = nofl1] LE_U (i_TO_SC(result_exp)comparison)[2], ozflag = oz EQ_U ZERO(input_exp)b_0*,

origin = re[1] LE U no[1], nz_plus_oz = re[1] ADD_U az,

 $pro_new_z = nz[2]$

 $pro_i po_i z = no[2]$

shift_add_sel = CASE DF1(t_oclave){ck,ocls}

#delay octs to match pipelin delay#

```
#detay octs to match pipelin detay#
                                                                nz_plus_oz[1..imput_exp+3],
b"0"CONC rz_plus_oz[1..input_exp+2],
b"00"CONC rz_plus_oz[1..imput_exp+1]
                                                                                                                                                                b"000"CONC nz_plus_oz[1..input_exp]
                                shift_add= MUX_4{STRING[input_exp+3]bit}
#keep 13 bits here to match no, keep msb's#
                                                                                                                                                                                                shift add sel
```

oct/3:quatro

ESAC,

oct/1:dos, oct/2:tres, JOIN (ck,reset,qshift,(iN_TO_S(input_exp)new)[2]) ->riz, (load_flags,(nzflag,origin,noflag,ozflag,motion,pro_new_z,pro_no_z))->flags, (ck,reset,qshift,CAST{ STRING[input_exp+1]bit}n_o)->no,

 $nz_r = (SC_TO_1\{12\} nz[1])[2],$ $no_r = (SC_TO_1\{13\} mo[1])[2],$ $oz_r = (SC_TO_1\{13\} oz)[2],$ $sa_r = (SC_TO_1\{13\} shift_add)[2].$

motion = shift_add LE_U no[1],

#value for simulation#

(ck, reset, (IN_TO_S(input_exp)old)[2]) ->az.

OUTPUT flags

#the buffer for the FIFO#

FN PULSE = (bool:ck,t_reset:reset,t_load:in) ->t_load: #a pulse generator, glitch free#

CASE mag_out #length of inputoded word#

OF input/0:b'00001",
input/2:b'00100",
input/3:b'00110",
input/3:b'00111",
input/5:b'00111",
input/6:b'01100",
input/(7..21):b'01100",

ELSE b'10000",
input/(22..37):b'1000"#

->STRING[5]bit

#the length of the huffman encoded word#

FN LENGTH = (input:mag out)

CASE (in,DFF{t_load}(ck,reset,in,read))
OF (write,read):write

ELSE ESAC. FN REV_BITS = (STRING(8)bit:in) ->STRING(8)bit:CAST(STRING(8)bit)(in[8],in[7],in[6],in[4],in[4],in[2],in[1]). FN FIFO_BUFFER = (bootck, t_resetreset,t_direction:direction,t_cycle:cycle,t_mode:mode, t_input:value mag_out_huft, STRING[16]bit:fito_in,t_fito:fito_full fito_empty.
STRING[32]bit:shift,STRING[2]bit:token_length, boot:flush_buffer,t_quant:tof_quant)

->(STRING[16]bit,STRING[16]bit,STRING[16]bit,STRING[5]bit,t_load,t_load]: #filo_out, s, filo_read filo_write#

BEGIN MAKEDFF_INIT{STRING[16]bit}:low_word high_word,

```
DFF_INIT(I high low):high low,
MUX_2{STRING[16]bil}:high in low_in high out low_out.
DFF INIT(STRING(5)bil):s,
```

CASE direction forward:left ELSE right ESAC, dir sa 回

token_cycle:b*000* CONC token_length, length = CASE cycle

ESAC

ESAC,

forward:b⁻⁰ CONC s[2.5] select_s = CASE direction ELSE s

ESAC,

new_s = (ADD_US_ACTEL(select_s,length,b'1))[2..6], #if new s pointer >16# #on inverse passed first 16 bits, active from [16,31] #

high_low_flag = new_s GE_U b*10000*.

ffo_not_full = CASE fifo_full
OF ok_fifo: write

#forward#

#flush buffer when frame finished# fifo_write = CASE high_low #type change# CASE flush_buffer OF hightwrite ELSE CASE

OF twite #needs 2 cycles to clear# ELSE CASE DFF{bool}(ck,reset,flush_buffer,f) OF twile

OF twrite ELSE read ESAC

data_ready = CASE fifo_emph OF ok_fifo:write ELSE ESAC,

#fload low on reset to start things# = CASE reset OF rstwite load_low

rst:write, no_rst: PULSE(ck,reset, CASE (high_low_flag,data_ready) #foed low word#

ELSE read ESAC)

#from inverse#

#delay reset for s and load_high# reset_s = DFF(I_reset)(ck,reset_reset,rest), **ELSE** read ESAC,

no_rst: PULSE(ck,reset, CASE (high_low_flag,data_ready) #load high word# =CASE reset_s #load high next# rstwitle, load high

OF (f,wite):wite ELSE read

ESAC)

ELSE read

ESAC,

#read control for data_in FIFO# = CASE load_low OF write:read fifo_read

ELSE CASE bad high

OF writeread ELSE write ESAC

#control signals#

(write_low,write_high) =CASE direction OF forward:[2]fifo_not_f ELSE ESAC,

forward.CASE high_low (high_out_sel,low_out_sel) = CASE direction OF forward:CASE High I

OF high:(left,right) ELSE (right,left) ESAC

ELSE [2]CAST(L_mud(s GE_U b-10000") ESAC.

(shift[17.32],fife_in,dir_sel)

(shift[1..16],ffo_in,dir_sel)

(high_word,low_word,high_out_sel)

->low_out.

(low_word,high_word,low_out_sell)

(ck, reset, write_low, low_in, ZERO(16]b*0") ->low_word,

->high_word, (ck,reset,write_high,high_in,ZERO(16)b"0")

(ck,resat,fifo_not_fuil,CASE high_low_flag OF_thigh ELSE low ESAC, low)

(ck, CASE forward

OFforward:reset

OF forward:fife not full ELSE reset s ESAC,CASE direction

new_s,ZERO(5)b"0") ->s. ELSE data_ready ESAC,

OUTPUT (low_word,low_out,high_out,s,fifo_read,fifo_write)

#the HUFFMAN decode/encode function#

#a pulse generator, glitch free#

FN PULSE = (bool:ck,t_reset:reset,t_load:in) ->t_boad: CASE (in,DFF(t_load)(ck,reset,in,read))

(write, read):write я Б

ELSE read

ESAC.

->STRING[16]bit FN SHIFT32_16 = (STRING[32]bit:buffer,STRING[5]bit:s) #left justified value, s shift const#

flinput values rotated so always shift<16# LET shift = (s AND_B 6011117)[2..5].

CAST{STRING[16]bit}{[INT j=1..16] MX16(CAST{STRING[16]bit}{[INT i=1..16]buffer[j-1+i]),shift) } OUTPUT

FN SHIFT16X16_32 = (STRING[16]bit:o n, STRING[4]bit:sel) ->STRING[32]bit:

LET sel_mux4= CASE sel[1..2] OFb'00":sel[3..4] ELSE b'11"

sel_mux4_high = CASE sel[1..2]

b*11*:sel[3..4]

8

ELSE ESAC, sel mux8 = CASE self1

OF b'0: sel[2..4] ELSE b"111"

ESAC,

MUX_8{bit}{n[4],n[3],n[2],n[1],o[4],o[4],o[4],c[4],CAST{[3]bool}sel_mtx8}, MUX_8{bit}{n[5],n[4],n[3],n[2],n[1],o[5],o[5],o[5],c[5],CAST{[3]bool}sel_mtx8}, MUX_8{bit}{n[6],n[5],n[4],n[3],n[2],n[1],o[6],o[6],CAST{[3]bool}sel_mtx8}, MUX_8{bit}{n[7],n[6],n[5],n[4],n[3],n[2],n[1],o[7],CAST{[3]bool}sel_mtx8},

MX16(CAST{STRING[B]bit}([[INT i=1.8]n[9-i])) CONC ALL_SAME[8]B_TO_S o[8],sel[1.4]), MX16(CAST{STRING[9]bit}([INT i=1..9]n[10-i]) CONC ALL_SAME[7]B_TO_S o[9],sel[1..4]), MX16(CAST{STRING[10]bit}([INT i=1..10]n[11-i]) CONC ALL_SAME[6]B_TO_S o[10],sel[1..4]), MX16(CAST{STRING[11]bit}([INT i=1..11]n[12-i]) CONC ALL_SAME[5]B_TO_S o[11],sel[1..4]),

MX16(CAST{STRING[12]bit}{[INT i=1..12]n[13-i]) CONC ALL_SAME{4}B_TO_S o[12],sel[1..4]), MX16(CAST{STRING[13]bit}{[INT i=1..13]n[14-i]) CONC ALL_SAME{3}B_TO_S o[13],sel[1..4]), MX16(CAST{STRING[14]bit}{[INT i=1..14]n[15-i])CONC ALL_SAME{2}B_TO_S o[14],sel[1..4]),

MX16(CAST{STRING[16]bit}(([INT i=1..15]n[16-i])CONC o[15]),sel[1..4]),

MX16(CAST{STRING[16]bit}{[iNT l=1..16]n[17-i]).sel[1..4])

MX16(CAST{STRING[16]bit}{bb CONC ([INT i=1...15]n[17-i]),sel[1..4]), MX16(ZERO{2}b'o* CONC CAST{STRING[14]bit}{[INT i=1...14]n[17-i]),sel[1..4]), MX16(ZERO{3}b'o* CONC CAST{STRING[13]bit}{[INT i=1..13]n[17-i]),sel[1..4]), MX16(ZERO{4}b'o* CONC CAST{STRING[12]bit}{[INT i=1..12]n[17-i]),sel[1..4]), MX16(ZERO{5}b*0* CONC CAST{STRING[11]bit}([iNT i=1..11]n[17-i]),sal[1..4]), MX16(ZERO{6}b*0* CONC CAST{STRING[10]bit}([iNT i=1..10]n[17-i]),sal[1..4]), MX16(ZERO{7}b*0* CONC CAST{STRING[9]bit}([iNT i=1..9]n[17-i]),sal[1..4]), MX16(ZERO{9}b*0* CONC CAST{STRING[9]bit}([iNT i=1..8]n[17-i]),sal[1..4]), MUX_8{bit}{b'0,n[16],n[15],n[14],n[13],n[12],n[11],n[10],CAST{[3]bool}sel_mux8_high),
MUX_8{bit}{b'0,b'0,n[16],n[15],n[14],n[12],n[11],CAST{[3]bool}sel_mux8_high),
MUX_8{bit}{b'0,b'0,b'0,n[16],n[15],n[14],n[13],n[12],CAST{[3]bool}sel_mux8_high),
MUX_8{bit}{b'0,b'0,b'0,b'0,n[16],n[15],n[14],n[13],CAST{[3]bool}sel_mux8_high),

MX_4{bit}{b'0,n[16],n[15],n[14],CAST{[2]bool}sel_mux4_high}, MX_4{bit}{b'0,b'0,n[16],n[15],CAST{[2]bool}sel_mux4_high}, MX_4{bit}{b'0,b'0,b'0,n[16],CAST{[2]bool}sel_mux4_high},

6,

_

EN.

->STRING[4]bit:CAST(STRING[4]bit)((in[4],in[3],in[2],in[1])) MAC REV_4 = (STRING[4]bit:in)

FN HUFFMAN_DECODE =(i_mode:mode,STRING[2]bit:token_length_in,STRING[32]bit:buffer,STRING[5]bit:s) #in is data from bus, fife_empty is input fife control#

```
->(bit,t_input,STRING[2]bit#token#):
```

BEGIN

mag_out2 = CASE input_decode(9..12]
OFb*1111*:(input_decode(13..16] ADD_U b*10110*)#add 22 to give value#
ELSE input_decode(9..12] ADD_U b*00111* #add 7 to give value# MAKE SHIFT32_16:input_decode. ESAC, EI RO 回

ESAC, ELSET

fadd 22 to give value# #add 7 to give value# mag_out2 = CASE sel_9_12 OF_t:REV_4 input_decode[13..16] ELSE_REV_4 input_decode[9..12] ESAC_ADD_U CASE sel 9 12 OFt: b*10110* ELSE **b***00111* ESAC,

ELSE CASE Input_decode[5] mag_out_huff=CASE input_decode[1] ELSE CASE Input_decode[4] b'1thput/3 ELSE CASE input_decode(3) OFb'1:input/2 OFb'1:input/1 OFb'0:input/0

```
OF IDE_Still: (S_TO_IN (CAST{STRING[9]bit}][NT j=1..9]input_decode[11-i]))[2]
                                                                                                                                                                                                                                                                                                                                                                                                                                                                            #select huff value, O(in tpf_send) or real value, rearange the bits for real data#
                                                                                                               ELSE (S_TO_IN (b*0000* CONC mag_out2))[2] ESAC
                                                                          ELSE CASE input_decode[8]
OFb'1.input/8
ELSE CASE input_decode[6]
                                      ELSE CASE input_decode[7]
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             fron for still bit 1 is sign bit so discard#
                                                          OFb'1:input/5
                    OF b'1:input/4
                                                                                                                                                                                                                                                                                                                                      OFlpf_still:input_decode[1]
ELSE CASE mag_out_huff
                                                                                                                                                                                                                                                                                                                                                                          OFinput/0:b'0
ELSE input_decode[2]
                                                                                                                                                                                                                                                                                                #on tpf_still bit 1 is the sign bit#
                                                                                                                                                      ESAC
                                                                                                                                                       ESAC
ESAC
ESAC
ESAC
ESAC
ESAC
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 mag_out = CASE mode
                                                                                                                                                                                                                                                                                                                sign = CASE mode
                                                                                                                                                                                                                                                                                                                                                                                                                  ESAC
ESAC,
```

token_tength = b*000*CONC token_length_in,

#decode token, valid only during a token cycle# token = CASE token_length[4..5]
OFb*10*:input_decode[1..2],
b*01*:input_decode[1] CONC b*0* ESAC.

JOIN (buffer,s) ->input_decode.

OUTPUT (sign, mag_out, token) ENO. #the huffman encoder#

FN HUFFMAN_ENCODE = (t_input:value,bit:sign,STRING[2]bit:token,t_mode:mode, t_cycle:cycle, STRING[16]bit:buffer,STRING[5]bit:s)

->(STRING[32]bit)

MAKE SHIFT16X16_32-shift. BEGIN

#encode value#

LET header = CAST(STRING[2]bil)(b'1,sign),

value_bit = CAST[[16]bit](IN_TO_S[16] value)[2],

sub_const = CASE value OF input/(7..21):b'00111*, input/(22..37):b*10110* ELSE b*00000*

sub_value = ((IN_TO_S{input_exp}value)[2] SUB_U sub_const)[8..11],

enc_value=

CASE cycle

OF token_cycle:token CONC ZERO(14)b"0", #token is msb, max 2 bits#

data cycle: CASE mode

OF IPE Still: CAST (STRING[1] bits sign CONC CAST (STRING[15] bits ([INT]=1..15] value bit[17-i]] #on intra & LPF pass thro value as 16 bit word, and reverse bit order, place sign first next to Isb#

#otherwise value is to Huffman encoded, so out 16 bit as this is the max, the shift removes the extra bits#

CASE value

input/0:b"0"CONC ZERO(15)b"0",

Input/1:header CONC b"1"CONC ZERO(13\b"0"

input/3:header CONC b 001 CONC ZERO(11)b 0 Input/2:header CONC b-01 CONC ZERO(12/b-0*

input/4:header CONC b 0001 CONC ZERO(10]b 0

nput/5:header CONC b 00001 CONC ZERO(9)b 0". nput/6:header CONC b 000001 CONC ZERO(8)b 0 input/(7..21);header CONC b*000000* CONC(REV_4 sub_value)CONC ZERO(4)b*0*, #sub 7 to give value#

#sub 22 to give value input/(22..37):header CONC b*000001111* CONC (REV_4 sub_value)

ELSE header CONC b-00000011111111

ESAC

ESAC,

#dummy value# skip_cycle:ZERO{16}b*0*

->shift. JOIN (buffer ,enc_value,s[2..5])

#max value is 37 so 8 bits enough#OUTPUT shift END.

some basic macros for the convolver, assume these will# #be synthesised into leaf cells#

MAC_MX_4[TYPE ty]=(ty:in1 in2 in3 in4, [2]bool:sel) CASE_sel

CASE OF (1

ESAC.

MAC ENCODE4_Z = (1_mux4:in)
CASE in
OF uno:(f,f),

uno:(f,f), dos:(f,f), tres:(f,f), quatro:(f,f)

ESAC.

MAC ENCODE3_2 = (t_muc3:in) CASE in OF !:(f,f), c:(f,f),

```
MAC MUX_3(TYPE t)=(t:in1 ln2 ln3, t_mux3:sel) ->t: MX_4(t)(in1,in2,in3,in1,ENCODE3_2 sel).
```

Ķ MAC MUX_4(TYPE t}=(l:in1 in2 in3 in4, t_mux4:sel) MX_4(t)(in1,in2,in3,in4,ENCODE4_2 sel).

MAC_MUX_2{TYPE I}=(I:In1 in2, 1_mucsel) ->t:

CASE sel

OF left:in1,

right:in2 ESAC.

MAC_MUX_B(TYPE ty)=(ty:in1 in2 in3 in4 in5 in6 in7 in8, [3]bool:sel) ->ty:

(f.f.1):in 1. CASE sel

(f,f,t):in2, (f,t,f):in3,

(1,1,1):In4,

(1.(1):in5, (1.(1):in6, (1.(1):in7, (1.(1):in8 ESAC.

MAC_MX16=(STRING[16]bit.in, STRING[4]bit.sel) ->bit.

CASE sel OF b*0000*:in[1], b*0001*:in[2],

```
MAC MX16=(STRING[16]bit:n, STRING[4]bit:sel) ->bit:

MUX_2{bit}{
    MUX_2{bit
```

b*1110*:in[15]. b*1111*:in[16]

6 1011 :in[12] b 1100 :in[13] b 1101 :in[14]

b*1001*:in[10] b*1010*:in[11]

b'0010':in[4], b'0011':in[4], b'0100':in[5], b'0110':in[7], b'0111':in[8], b'1000':in[9],

quant/4:(1,1,0), quant/5:(1,1,1), quant/6:(1,1,0), quant/7:(1,1,1) ESAC. quant/3:((1,1,1)

COM MAC MUX_3{TYPE I}=(tin1 in2 in3, t_mux3:sel) ->t: CASE sel

OFI:in1, c:in2,

r.in3

MAC MUX_4[TYPE t]=(t:in1 in2 in3 in4, t_mux4:sel) ->t: CASE sel

OFuno:in1,

quatro:in4 tres:in3, dos:in2,

ESAC.

FN NOT = (boot:in)->boot:CASE in OF tf,ft ESAC.

FN XOR = (boot: a b) ->bod: CASE (a,b) OF (1,1)|(t,t):1 ELSE 1 ESAC.

FN AND = (boot: a b) ->bool: CASE (a,b) OF (t,t):t,

(i,bool)](bool,f):f ESAC.

FN OR = (bool: a b) ->boot: CASE (a,b)
OF (f,f):f,
(i,bool)|(bool,t):1
ESAC. MAC DEL(TYPE i) ->t:DELAY(?t,1).

#a general d latch# MAC LATCH {TYPE t}=(t_load:load,t:in) ->t: BEGIN

LET out-CASE load MAKE DEL(I):del.

ELSE del ESAC.

JOIN out->del. OUTPUT out

#a general dff# MAC DF1 (TYPE tj=(bool:ck,t:in) ->t: BEGIN

MAKE DEL (1):del.

OUTPUT del

JO!N in->del.

#a resetable DFF, init value is input parameter# MAC DFF_INIT(TYPE t)=(boot:ck,t_reset:reset,t_load:load,t:in init_value) ->t

BEGIN

MAKE DEL(I):del.

LET out=CASE (load,reset)

OF (write,t_reset):in, (read,rst):init_value

ELSE del

ESAC.

OUTPUT CASE reset JOIN out->del.

OF rst:init value

ELSE dei **ESAC**

END.

#a dff resetable non-loadable dff#

MAC DFF(TYPE t)=(bool:ck,t_reset:reset,t:in init_value) ->t: BEGIN

MAKE DEL(I):del.

JOIN in->def.

OF rst.init_value ELSE_del ESAC **OUTPUT CASE reset**

```
MAC PDEL{TYPE t,INT n} = (t:in) ->t:

IF n=0 THEN DEL(t)in

ELSE PDEL(t,n-1) DEL(t) in

F1.
```

MAC PDF1{TYPE t,INT n} = (bool:ck,t:in) ->t:
IF n=0 THEN DF1{}(ck,in)
ELSE PDF1{t,n-1}(ck,DF1{t}(ck,in))

#generates the new_mode from the old, and outputs control signals to the tokeniser#

STRING[2]bit:token_in,t_octave:octave,t_state:state,t_direction:direction,t_load:load_mode_in FN MODE CONTROL = (bool:ck, t_reset:reset, t_intra:intra_inter,bool:lpf_done,[7]bool:flags, ,t_cycle:cycle)

->(i_mode,i_mode,t_mode,STRING[2]bit,t_diff,STRING[2]bit,t_mode):
#new_mode,proposed mode,current token,difference,token_length, #

BEGIN

MAKE [4]DFF INIT(I_mode):mode,
DFF INIT(I_diff):diff_out,
DFF INIT(I_mode):next_mode.
LET nzflag=flags[1],

ortgin=flags[2], noflag=flags[3], ozflag=flags[4], motion=flags[5], pro_new_z = flags[6], pro_no_z = flags[6], #synchronise mode change at end of LPF# lpf_done_del = DFF{bool}(ck,reset,lpf_done,f).

LET next = (SEQ

#the proposed value for the mode at that octave, flags etc will change this value as necessary# #proposed, or inherited mode from previous tree#

pro_mode:= CASE reset

#reset on frame start, so do lpf# OF rst:CASE infra_inter

OF intra:tof_still ELSE tof_send ESAC

ELSE CASE lpf_done_del

OFt:CASE intra inter #store default mode in mode[4]#

OF intra:still

ELSE send ESAC

ELSE CASE state

OFdown1:mode[3], #jump sideways in oct/1#

, up0:mode[4]

OF oct/0:mode[1] ELSE CASE octave

oct/1:mode[2], oct/2:mode[3]

ESAC

ESAC

ESAC,

#inhert the previous mode# new_mode:=pro_mode, token_out:=b*00*,

OF t:new_mode:=stop CASE pro_mode OFvoid:CASE ozflag ELSE ESAC,

token length:=b*00*, flag:=f; CASE direction OF forward:

difference:=nodiff,

#stay in these modes until end of tree#

#intra so must zero out all of tree#

void_still:,

OF tinew_mode:=stop ELSE new_mode:=void still_send:(token_length:=b*01*; CASE (nzflag OR pro_new_z) OF t(token_out:=b*00*; CASE ozflag

new_mode:≂still_send) ESAC ELSE (token_out:=b*10*; **ESAC**

send: CASE ozflag OFt:(token_length:=b*01*;

CASE (nzflag OR pro_new_z)

OF t:(token out:=b*00*;

new mode:=stop)

new_mode:=stiff_send ESAC

ESAC ESAC

still: (token length:=b"01";

CASE (nzflag OR pro_new_z) OFt: (token out:=b*00*; new mode:=vold_still) #zero out tree# ELSE (token_out:=b*10*;

new_mode:=still

ESAC

#for ELLA only DUMBII#

(pf_still):(token_out:=b'00"; token_length:=b'00"),

(lpf_send):(difference:=diff,

token_length:=b'01";

CASE (noffagi OR pro no z

new_mode:=tpf_send) #as mode stop but for this block only# ELSE (token_out:=b*10*; ESAC)

ESAC,

inverse: CASE pro_mode

SUBSTITUTE SHEET (RULE 26)

ELSE CASE (octave, load_mode_in)

ELSE new_mode:=void

ESAC

(oct/1,write):(write,write,read,read), (oct/2,write):(read,write,write,read) ELSE (read, read, read, read) P.

#save the new mode& difference during a token cycle, when the flags and tokens are valid# (ck,reset,CASE cycle

OF token_cycle:write

ELSEread

->next mode, ESAC, next[1],still)

token_cycle:write (ck,reset,CASE cycle ELSE read

ESAC, next[4],nodiff)

->diff out.

#now write the new mode value into the mode stack at end of cycle, for later use # FOR INT i =1..4 JOIN (ck,no_rst,load_mode[i],CASE (reset,lpf_done_del)
OF (no_rst,t)[(rst,bool):next[2]
ELSE next_mode
ESAC,stiil) ->mode[i].

#dont update modes at tree base from lpf data, on reset next[1] is undefined#

OUTPUT (next_mode,next[2],next[3],diff_out,next[5],next[1])

#threshold = 2"quant_norm# #the tree coder chip#

FN PALMAS= (boot:ck,t_reset:reset,t_direction:direction,t_intra:intra_inter,t_channel_factor.channel_factor,

[4]t_quant:quant_norm, STRING[16]bit:buffer_in, t_input:new old,[4]t_result:threshold, t_fifo:fifo_full fifo_empty, STRING[xsize]bit:col_length, STRING[ysize]bit:yimage_string,STRING[11]bit:yimage_string_3#yimage& yimage*2.5#) STRING[ystze]bit.row_length,STRING[xsize]bit.ximage_string,#ximage#

->(i_input,t_sparc_addr,(t_load,t_cs),(t_load,t_cs),STRING[16]bit,[2]t_load,bool,t_cyde):

#old,address,(rw_new,cs_new),(rw_old,cs_old),buffer_out,fifo_read fifo_write, cycle#

BEGIN

MAKE DECIDE: decide,

ADDR GEN:addr gen,

HUFFMAN ENCODE:huffman encode,

FIFO BUFFER: fifo buffer,

HUFFMAN DECODE:huffman_decode,

CONTROL_COUNTER:control_counter, BLK_SUB_COUNT: sub_count, MODE CONTROL mode,

DFF_INIT(i_channel):channel QUANT:quant.

E

motion=decide[5] nzflag=decide[1] noflag=decide[3] ozflag=decide[4] origin=decide[2],

pro no z = decide[7],#pro no z or pro new z# pro_new_z = decide[6],

```
new_mode = mode[1],
pro_mode = mode[2],
token_out = mode[3],
difference = mode[4],
token_length = mode[5],
pro =quant[1], #pro_no, or pro_new#
lev_out = (S_TO_iN quant[2])[2],#corresponding level#
sign = quant[3], #and sign #
```

```
isy_out = (S_10_in quam(2))[2], #corresposign = quant[3], #and sign #

octs = addr_gen[2],

sub_en = addr_gen[3],

tree_done = addr_gen[3],

ipf_done = addr_gen[5],

state = addr_gen[6],

cycle =control_counter[2],

cs_new=control_counter[7],

rw_new=read,

rw_otd=control_counter[8],

cs_old=control_counter[8],
```

```
load_channel≈ CASE (sub_en,sub_count[2]) #change channel#
OF(t,t):write
ELSE read
ESAC,
```

new_channel = CASE channel_factor
OF huninance:y
ELSE CASE channel
OF y:u,

flush_buffer =DFF{bool}(ck,reset,CASE channel_factor OFluminance:CASE load_channel #flush the buffer in the huffman encoder# ur, vry ESAC ESAC,

OF write:

ELSE 1 ESAC,

color: CASE (channel, load_channel)

OF(v,write):t ELSE f

ESAC

frame_done = PDF1{bool,1}(ck,flush_buffer),

fito_write=fito_buffer[6], fito_read =fito_buffer[5], s =fito_buffer[4],

buffer_out = fifo_buffer[1],

lev_in = huffman_decode[2], sign_in = huffman_decode[1], token_in = huffman_decode[3],

del_new = PDF1{(_input,4}(ck,new),

```
OFlpf_still|pf_send|pf_stop:quatro
ELSE CASE (octs,channel)
                 del_old = CASE (direction,pro_mode)
                                                                                                                                                                                                                                                             (oct/1,y)|(oct/0,u|v):dos,
#old has variable delays for inverse#
                                                                                                                           ELSE control_counter[3]
                                                                                                                                                                                  oct_sel = CASE pro_mode
                                                                                                                                                                                                                                         OF (oct/0,y):uno,
                                                                                       decide_reset=CASE reset
                                                                                                         OF rstrst
                                                                                                                                            ESAC,
```

JOIN (ck,decide_reset,threshold_act,new,old,threshold_act,threshold_act,acts,control_counter[6])->decide, quant_oct = MUX_4(t_quant)(quant_norm[1],quant_norm[2],quant_norm[3],quant_norm[4],oct_sel) (ck, reset, intra_inter, lpf_done, decide, token_in, octs, state, direction, control_counter[1], cycle)-> mode,

threshold_oct = MUX_4(i_result)(threshold[1],threshold[2],threshold[3],threshold[4],oct_sel),

#delay the new&old values by 5 or 1 depending on mode & direction#
((IN_TO_S(input_exp)del_new)[2], (IN_TO_S(input_exp)del_old)[2],
(IN_TO_S(input_exp)lev_in)[2], sign_in,direction,quant_oct,difference.pro_mode) ->quant,

(oct/2,y)|(oct/1,u|v):tres

ESAC

```
ximage_string.yimage_string.yimage_string_3,control_counter[4],control_counter[5],new_mode)->addr_gen,
(ck, reset , new_channel, channel, toad_channel, sub_count[1], col_length, row_length,
```

->fito buffer, (ck, reset, direction, cycle, pro_mode, lev_out, huffman_decode [2], buffer_in, fifo_full, fito_empty,huffman_encode, token_length, flush_buffer,quant_norm[4])

->huffman_decode, ->huffman encode, (pro_mode,token_length,fifo_buffer[2] CONC fifo_buffer[3],fifo_buffer[4]) (lev_out, sign, token_out, pro_mode, cycle, fifo_buffer [2], s)

->sub_count, (ck,reset,sub_en,t,t)

->control_counter,

->channel. (ck,reset,pro_mode,new_mode,direction) (ck,reset,load_channel,new_channel,y)

(CASE new mode

OF void|void still:input/0 ELSE ESAC

(S_TO_INpro)[2] ,addr_gen[1],(rw_new,cs_new),(rw_old,cs_old),buffer_out,(fffo_read,fifo_write),frame_done,cycle)

88

#the decoder for the barrel shifter- decides if the bit value and q value are in the upper-triangle, or diagonal and set the control bits ->[qmax](bool#upper diag#,bool#diagonal#): MAC DECODE(INT n) = (i_quant:q)

#one bit of the decoder#

MAC DECODE_BIT(INT []= (t_quantq) ->(bool,bool):

OF quant/(0..qmax-j):(t,f), #upper triangle#

quant/(qmax-j+1):(f,t) #diagonal#

ELSE (f,1)

OUTPUT((INT J=1..qmax]DECODE_BIT{[](q))

```
->(bit,bit):#level[],round_level[]#
#now the selector fn to mux between the data in bit ,0 or 1 depending on q#
                                                 ->(STRING[n]bit#level#,STRING[n]bit#round_level#):
                                                                                                                                                                                                                                                                                                                                                                                                     ->select[]
                                                                                                                                   MAC SELECT_BIT = ([2]bool:upper_or_diag,bit:data)
                     MAC SELECTOR = (L_quant:q,STRING[INT n]bit:data)
                                                                                                                                                                                                                                                                                                                                                                                                     FOR INT j=1..qmax JOIN (decode[]],data[n-qmax+]]
                                                                                                                                                                                       #upper-triangle#
                                                                                                                                                                                                                                           #lower-triangle#
                                                                                                                                                                                                                 #diagonal#
                                                                                                                                                                                                                                                                                                                     [qmax]SELECT_BIT: select.
(q) ->decode.
                                                                                                                                                                                                                                                                                          MAKE DECODE(n):decode,
                                                                                                                                                          CASE upper or diag
OF (t,f):(data,data),
                                                                                                        #the 3->2 bit selector#
                                                                                                                                                                                                              (f,t):(b'0,b'0)
                                                                                                                                                                                                                                         ELSE (60,6'1)
                                                                                                                                                                                                                                                                    ESAC.
                                                                               BEGIN
```

#now the selector fn to shift the level depending on q#

MOC.

OUTPUT (data[1..n-qmax] CONC (BIT_STRING(qmax}([INT j=1..(qmax)]select[][1])), #level# data[1..n-qmax] CONC (BIT_STRING(qmax]([INT j=1..(qmax)]select[][2])) #round_level#

MAC BARREL_SHIFT_RIGHT = (t_quant:q,STRING[INT n]bit:data) ->(STRING[n]bit#level#): MUX_8[STRING[n]bit](

b"0"CONC data[1..n-1]

b*00*CONC data[1..n-2]

b*000*CONC data[1..n-3]

b*0000*CONC data[1.n-4]

b'000000"CONC data[1..n-6], b'00000'CONC data[1..n-5]

b"0000000"CONC data[1..n-7],

NT_BOOL 4).

#the bshift for the inverse, to generate the rounded level #

MAC BARREL_SHIFT_LEFT = (1_quant:q,STRING[INT n]bit:data#lev#) ->(STRING[n]bit#round_level#): MUX_8(STRING[n]bit](

data[2.n]CONCb*0*,

data[4.n]CONCb*011 data[3..n]CONCb*01

data[5.n]CONCb*0111

data[7..n]CONCb*011111* data[6..n]CONCb*01111

data[8..n]CONCb*0111111 INT_BOOL q). #the function to return the quantised level(UNSIGNED), and proposed value given,# # the new&old values, forw/inverse direction # FN QUANT = (STRINGlinput_explbit: new old lev_inv,bit:sign_lev_inv, t_direction.t_quant:q,t_diff.diff.cence,

-> (STRING[input_exp]bit,STRING[input_exp]bit,bit) #pro,lev& sign#:

BEGIN

#decide which of new-old or new will be quantised, and the sign of the level# #level is stored in sign &magnitude form#

OF forward:left, inverse:right dir_sel = CASE direction ESAC,

#put old=0# sub_sel = CASE difference OF diffielt ELSE right ESAC, sub_in= MUX_2{STRING[input_exp]bit}{old,ZERO{input_exp}b^0,sub_sel},

no =ADD_SUB_ST(new,sub_in,subt),

lev_final= ABS_S no, #now input_exp+1 bits#

sgn_level = MUX_2(bit)(#sign of value to be quantised#

no(1), sign_lev_inv, dir_sel).

#find the quant. level by shifting by q, for the inverse it comes from the Huffman decoder#

lev data = BARREL_SHIFT_RIGHT(q,lev_final),

#salurate the lev at 37, for the Huffman table, except in tof_still mode, sond all the bits#

lev_forw = CASE mode

OF tof still lev data
ELSE CASE lev data GT_U b*00000100101*

OFt:b'00000100101

ELSE lev_data

ESAC

ESAC,

lev = MUX_2(STRING[input_exp+1]bit]{

lev_forw, b"0" CONC lev_inv,

dir sel),

lev $z = \text{lev } EQ_U ZERO(\text{input} \text{ exp+1}b^*0^*$, #the level = 0 flag#

inv lev z = CASE lev z ELSEVI OFtbo

round lev = BARREL_SHIFT_LEFT(q,lev) AND_B CASE mode #the level value shifted up, and rounded#

OF Inf_still:b*00* CONC ALL_SAME(input_exp-1]b*1*
ELSE BIT_STRING(input_exp+1){(input_exp+1]inv_lev_z) ## lev=0 out all 0's#

ESAC, #dear out extra bit for tpf_still case# #calculate the proposed value:in the case n-o round_lev is unsigned 10 bit, so result needs 11 bits# #pro_no will always be in range as round_lev<[n-o]

pro_no = ADD_SUB_ST(old,round_lev,CASE sgn_level OFbV:add, b1:subt ESAC),

#now pro_new = +/- round_lev#

round_sel = CASE sgn_level OFb0: left, b'1: right ESAC,

pro_new = MUX_2{STRING[input_exp+1]bit}(
round_lev ,
(NEG_U round_lev)[2..input_exp+2], #NEG sign extends#
round_set],

out_sel = CASE difference OF diffieft ELSE right ESAC. OUTPUT (MUX_2(STRING[input_exp]bit)(

pro_no[3..input_exp+2], pro_new[2..input_exp+1] out_sel), lev(2..input_exp+1), egn_level) #actel 1 bit full adder with active low cin and could ->(bit,bit):#cob,s# FN FA1B = (bit: ain bin cinb)

LET a c = B_TO_S ain CONC NOT_B(B_TO_S cinb), b_c = B_TO_S bin CONC NOT_B(B_TO_S cinb), out = ADD_U(a_c,b_c).

OUTPUT(CAST{bit} NOT_B(B_TO_S out[1]), out[2]) BEGIN

#a Ripple carry adder using 1 bit full adder blocks#

#the actel version of the ADD BIOP's#

MAC ADD S_ACTEL = (STRINGINT m)bit:ain,STRINGINT n|bit:bin,bit:cinb) ->STRINGIF m>=n THEN m+1 ELSE n+1 Fl|bit:

MAKE [IF m>=n THEN m ELSE n FIJFA18:sum.

LET a_c = IF m>=n THEN ain ELSE ALL_SAME(n-m)B_TO_Sain[1] CONC ain Fl. b_c = IF n>=m THEN bin ELSE ALL_SAME(m-n)B_TO_Sbin[1] CONC bin Fl. #signed nos so sign extend # LET subsignal = sum.

JOIN (a_c||Fm>=n THEN m ELSE n FI],b_c||Fm>=n THEN m ELSE n FI],cinb) ->sum||Fm>=n THEN m ELSE n FI]

->sum((IF m>=n THEN m ELSE n Ft) -j. JOIN (a_c[(IF m>=n THEN m ELSE n FI) -j],b_c[(IF m>=n THEN m ELSE n FI) -j]. sum[(IF m>=n THEN m ELSE n FI) -j+1[1]) FOR INT j=1..(IF m>=n THEN m ELSE n FI) -1

OUTPUT CAST{STRING[IF m>=n THEN m+1 ELSE n+1 Fl]bit]

MAC ADD_US_ACTEL = (STRING[INT m]bit:ain,STRING[INT n]bit:bin,bit:cinb) ->STRING[IF m>=n THEN m+1 ELSE n+1 F]|bit:

MAKE (IF m>=n THEN m ELSE n FIJFA1B:sum.

LET a c = IF m>=n THEN ain ELSE ZERO(n-m)b"0" CONC ain FI, b c = IF n>=m THEN bin ELSE ZERO(m-n)b"0" CONC bin FI. #unsigned nos so extend by 0# **LET subsignal = sum.**

JOIN (a_qiif m>=n THEN m ELSE n FI),b_qiif m>=n THEN m ELSE n FI),chrb) ->sumiif m>=n THEN m ELSE n Fi].

->sum((IF m>=n THEN m ELSE n Fl) 4]. JOIN (a_c((IF m>=n THEN m ELSE n FI) - J, b_c((IF m>=n THEN m ELSE n FI) - J, SUM[(IF M>=1) THEN M ELSE IN 1+1 [1]) FOR INT j=1..(IF m>=n THEN m ELSE n FI) -1

OUTPUT CAST(STRING(IF m>=n THEN m+1 ELSE n+1 Fijbt) (NOT_B(B_TO_S sum[1][1]) CONC

CAST(STRINGIP m>=n THEN m ELSE n FIJbit) (INT j=1...JF m>=n THEN m ELSE n FIJ sum[i][2])

END.

MAC ADD_SUB_ST =(STRING[INT m]bit:ain,STRING[INT n]bit:bin,t_add:sel] ->STRING[IF m>=n THEN m+1 ELSE n+1 FI]bit:

BEGIN

#sign extend inputs#

LET a s = CAST{STRING{1}bit}ain{1} CONC ain,
b s = CAST{STRING{1}bit}bin[1] CONC bin,
sel bit = CAST{STRING{1}bit}sel,
#ACTEL#
bin_inv = XOR_B(n+1){b_s, ALL_SAME{n+1}sel_bit},

#cinb is active low so cast sel(add->0,sub->1) & invert it#
out= ADD_S_ACTEL(a_s,bin_inv,CAST(bit)NOT_B sel_bit),
binout= out[2..1F m>=n THEN m+2 ELSE n+2 FI]

OUTPUT binout

END.

#transformation ops#
MAC B_TO_S= (bit:in) ->STRING(1)bit: CASE in
OF b'0:b"0",

MAC I TO SCIINT n) * (tresult: in) -> (flag,STRIING(n)bit): BIOP TRANSFORM S. MAC SC_TO_I(INT n) = (STRING(n)bit:in) -> (flag,t_result): BIOP TRANSFORM_S.

61:51

MAC S_TO_IN = (STRING[INT n]bit:in) -> (flag,t_input): BIOP TRANSFORM_S. MAC IN_TO_S(INT n) = (t_input: in) -> (flag,STRING[n]bit): BIOP TRANSFORM_S. MAC U_TO_IN = (STRING[INT n]bit.in) -> (flag,t_input): BIOP TRANSFORM_US.

MAC U TO LEN = (STRING(INT n]bit:in) -> (flag,t_length); BIOP TRANSFORM US. MAC LEN_TO_U(INT n} = (t_length:in) -> (flag,STRING[n]bit); BIOP TRANSFORM_US.

MAC O TO U(INT n) = (1 quant:in) -> (flag.STRING[n]bti): BIOP TRANSFORM US.

MAC S TO C = (STRING[INT n]bti:in) -> (flag.1_col):BIOP TRANSFORM US.

MAC S TO R = (STRING[INT n]bti:in) -> (flag.1_col):BIOP TRANSFORM US.

MAC S TO B = (STRING[INT n]bti:in) -> (flag.1_btk):BIOP TRANSFORM US.

MAC S TO SUB = (STRING[INT n]bti:in) -> (flag.1_sub):BIOP TRANSFORM US.

MAC S TO SPARC = (STRING[INT n]bti:in) -> (flag.1_sparc_addr):BIOP TRANSFORM US.

MAC S_TO_SPARC = (STRING[INT nipit:in) ->(flag,1_sparc_addr):BIOP INANSPORM MAC S_TO_SPARC = (STRING[INT nipit:in) -> (flag,STRING[nipit): BIOP TRANSFORM US.

MAC R_TO_S[INT nipit:in) -> (flag,STRING[nipit): BIOP TRANSFORM US.

MAC R_TO_S[INT nipit:in] -> (flag,STRING[nipit): BIOP TRANSFORM US.

MAC I_TO_Q = (I_input:in) ->1_quant:ARITH in.

MAC B_TO_!= (bit:in) -> result: CASE in OF bronesult/0, b'1:result/1

ESAC.

MAC CARRY= (t_add:in) ->STRING[1]bit: CASE in OF add:b*0*, subt:b*1*

ESAC. MAC BOOL_BIT = (bool:in) ->STRING[1] bit: CASE in

OF tb1

MAC BOOL_STRING(INT n) = (In)bookin) ->STRING(n) bit: (LET out = \overrightarrow{BOOL} _BIT in(1).

ELSE b'0'

ESAC.

OUTPUT IF n=1

ELSE ouf1] CONG BOOL_STRING[n-1] (n[2.n]) THEN OF

ㄸ

MAC BIT_STRING(INT n) = (Injbit.in) ->STRING(n) bit: (LET out = B_TO_S in[1].
OUTPUT IF n=1

ELSE out[1] CONC BIT_STRING(n-1)(in[2..n]) THENOG

MAC ZERO(INT n) = (STRING[1]bit:dummy) ->STRING[n]bit: IF n=1 THEN b*0* ELSE b'0' CONC ZERO(n-1) b'0' MAC ALL_SAME(INT n) = (STRING[1]bit:dummy) ->STRING[n]bit: IF n=1 THEN dummy ELSE dummy CONC ALL_SAME(n-1) dummy

MOS

The operators described in this section are optimal and take two-valued operands and produce a two-valued result. They may not be used with ELLA-integers or associated types.

The first basic value of any two-valued type declaration of the operand(s) and the result are interpreted by the operations as false, and the second basic value is interpreted as true. Thus, given the following type declarations:

MOC

MAC AND_T = (TYPE t. a b) -> t: BIOP AND.

MAC OR $T = (TYPEt: ab) \rightarrow t: BIOP OR.$

MAC XOR_T = $(TYPE t: a b) \rightarrow t: BIOP XOR.$

MAC NOT_T = (TYPE t: a) -> t: BIOP NOT.

MOC

The following operations take bit-string operand(s) and are bitwise, ie the operation is performed on the operand(s) one bit at a time. The operand(s) and result must all be ELLA-strings of the same length.

MAC AND_B = (STRING[INT n]bit,STRING[n]bit) -> STRING[n]bit: BIOP AND.

MAC OR_B = (STRING[INT n]bit,STRING[n]bit) -> STRING[n]bit: BIOP OR.

MAC XOR_B = (STRING[INT n]bit,STRING[n]bit) -> STRING[n]bit: BIOP XOR.

MAC NOT B = (STRINGINT njbit) -> STRING[njbit] BIOP NOT.

30

The operators described in this section may be used with primitive types le all enumerated types, except associated types, rows, strings and structures. These operations take two operands which must be of the same type and the result can be any two-valued type; we have packaged these BIOPs so they output a value of type 'bool' - you may change this if you wish.

MAC EQ = (TYPE t: a b) -> book BIOP EQ.

MAC GT = (TYPE t: a b) -> bool: BIOP GT.

MAC GE = (TYPE t: a b) -> bool: BIOP GE.

MAC LT = (TYPE t: a b) -> boot: BIOP LT.

MAC LE = (TYPE t: a b) -> bool: BIOP LE.

NOTE: these BIOPs are designed to take any primitive ELLA type. Since it is not possible to distinguish between primitive and other types, whilst leaving the macro declaration general enough to allow the use of all two-valued types that might be declared, there are type-checking limitations. This is done at network assembly, so use of illegal types will not generate an error

message until then.

NB: ARITH provides for relational operations on ELLA-integer types. MOC

Į,

These operations are optimal in their handling of ?? and operate on bit-string representations of unsigned integers. The result may be any two-valued type; we have used type 'boof'. The inputs can be of different lengths and different types.

MAC EQ_U = (STRING[INT n]bit,STRING[INT m]bit) -> boot: BIOP EQ_US.

MAC GT_U = (STRING[INT n]bit,STRING[INT m]bit) -> boot: BIOP GT_US.

MAC GE_U = (STRING[INT n]bit,STRING[INT m]bit) -> boot: BIOP GE_US.

MAC LT_U = (STRING[INT n]bk,STRING[INT m]bk) -> bool: BIOP LT_US.

MAC LE_U = (STRING[INT n]bit,STRING[INT m]bit) -> bool: BIOP LE_US.

Bit-strings representing signed numbers # COM

These operations are optimal and operate on bit-string representations of signed integers. The result may be any two-valued type; we have used type

'bool'. The inputs can be of different lengths and different types. Moc

MAC EQ_S = (STRING[INT n]bit,STRING[INT m]bit) -> book: BIOP EQ_S.

MAC GT_S = (STRING[INT n]bit,STRING[INT m]bit) -> bool: BIOP GT_S.

MAC GE_S = (STRING[INT n]bit,STRING[INT m]bit) -> boot: BIOP GE_S.

MAC LT_S = (STRING[INT n]bit,STRING[INT m]bit) -> bool: BIOP LT_S.

MAC LE_S = (STRING[INT n]bit,STRING[INT m]bit) -> boot: BIOP LE_S.

Shift operations

parameterised by the number of bits to be shifted (INT p). The macro and BIOP parameters must match. Note that no bits are lost in these shift operations, These operate on bit-strings. Both the enclosing macro and the BIOP are so you may need to trim the result to achieve the desired effect.

SR means shift right; SL means shift left.

The macros with the suffix 'S' perform arithmetic shifts; those with the

suffix '_U' perform bool shifts. MOC MAC SL_S(INT p) = (STRING[INT n|bit) -> STRING[n + p]bit: BIOP SL[p].

MAC SL_U(INT p) = (STRING(INT njbit) -> STRING(n+pjbit: BIOP SL(p).

MAC SR_S(INT p) = (STRING(INT n]bit) -> STRING(n + p]bit BIOP SR_S(p). MAC SR_U{INT p} = (STRING(INT n)bit) -> STRING(n + p|bit: BIOP SR_US(p).

Arithmetic operations

Bit-strings representing unsigned numbers

addition.

MAC ADD_U = (STRING[INT m]bit,STRING[INT n]bit)
-> STRING[IF m >= n THEN m+1 ELSE n+1 FI]bit:
BIOP PLUS US.

subtraction on bit-string representations of unsigned integers. Output is # signed.

MAC SUB_U = (STRING[INT m]bit,STRING[INT n]bit)
-> STRING[IF m >= n THEN m+1 ELSE n+1 Fi]bit;

BIOP MINUS US.

negation. Output is signed.

MAC NEG_U = (STRING[INT n]bit) -> STRING[n+1]bit: BIOP NEGATE_US.

multiplication.

MAC MULT_U = (STRING[INT m]bit,STRING[INT n]bit) -> STRING[m+n]bit: BIOP TIMES_US.

8

ok' and the second and third elements are the quotient and remainder; - divide. If the divisor is non-zero then the first element of the output is otherwise, the first element is 'error' and the rest is set to "?".

MAC DIV_U = (STRING[INT m]bit,STRING[INT n]bit) -> (flag,STRING[m]bit,STRING[n]bit):

BIOP DIVIDE US.

square root.

MAC SORT_U = (STRING[INT n]bit) -> STRING[(n+1) % 2]bit: BIOP SORT_US.

modulus (result always positive). If the divisor is non-zero, then the first element of the output is 'ok' and the second element is the modulus; otherwise, the first element is 'error' and the second is ??.

№

MAC MOD U = (STRING[INT m]bit, STRING[INT n]bit)

-> (flag,STRING[n]bit):

BIOP MOD US.

M03

 convert between one range of bit-string and another. If the input value cannot be represented as a legal value for the output string, the result is

error and '?'.

MOC

MAC RANGE_U (INT m) = (STRING[INT n]bit)

-> (flag,STRING[m]bit):

BIOP RANGE US.

Bit-strings representing signed numbers

addition.

MAC ADD_S = (STRING[INT m]bit;STRING[INT n]bit)

-> STRINGIF m >= n THEN m+1 ELSE n+1 FIbit:

BIOP PLUS_S.

subtraction.

MAC SUB_S = (STRING[INT mjbit,STRING[INT njbit)

-> STRINGIF m >= n THEN m+1 ELSE n+1 Filbit:

BIOP MINUS_S.

negation.

MAC NEG_S = (STRING[INT n]bit) -> STRING[n+1]bit: BIOP NEGATE_S.

multiplication.

MAC MULT $S = (STRING[INT m]bit, STRING[INT n]bit) -> STRING[m+n]bit: BIOP TIMES_S.$

NO.

divide. If the divisor is non-zero then the first element of the output is 'ok 'and the second and third elements are the quotient and remainder, otherwise, the first element is 'error' and the rest is set to '?'. The remainder has the same sign as the divisor.

၁၀

MAC DIV_S = (STRING[INT m]bit,STRING[INT n]bit)
-> (flag,STRING[m]bit,STRING[n]bit):
BIOP DIVIDE_S.

modulus (result always positive). If the divisor is non-zero, then the first element of the output is 'ok' and the second element is the unsigned modulus; otherwise, the first element is 'error' and the second is '?'.

MAC MOD_S = (STRING[INT m]bit,STRING[INT n]bit)
-> (flag,STRING[n]bit):

BIOP MOD_S.

∑

 convert between one range of bit-string and another. If the input value cannot be represented as a legal value for the output string, the result is 'error' and '?'.

MOC

MAC PANGE_S (INT m)= (STRING(INT n]bit)

-> (flag,STRING[m]bit):

BIOP RANGE_S.

absolute value. The output represents an unsigned integer.

MAC ABS_S = (STRING[INT n]tit) -> STRING[n]bit: BIOP ABS_S.

Built in Register

MAC DREG(INT interval delay) = (TYPE t) -> t: ALIEN REGISTER (interval, ?1, 0, delay).

MAC GEN_DREG(INT interval, CONST (TYPE I): Init, INT skew delay) = (t) -> t. ALIEN REGISTER (interval, init, skew, delay).

Built in type conversion

MAC CAST(TYPE t) = (TYPE 6) -> t: ALIEN CAST.

```
MAC ALL_SAME(INT n) = (STRING(1)bit:dummy) ->STRING(n]bit:
```

DEGIN

FAULT IF n < 1 THEN "N<1 in ALL_SAME" FI.

OUTPUT IF n=1 THEN dummy

ELSE dummy CONC ALL_SAME(n-1) dummy

E

S

MAC CAST {TYPE to} = (TYPE from:in) ->to:ALIEN CAST.

MAC ZERO(INT n) = (STRING[1]bit:dummy) ->STRING[n]bit:

FAULT IF n < 1 THEN "N<1 in ZERO" FI.

OUTPUT IF n=1 THEN 6'0"

ELSE b'0" CONC ZERO(n-1) b'0"

I

END.

MAC B_TO_S= (bit:in) ->STRING[1]bit: CASE in OF bb:b'0', b'1:b'1'

ESAC.

MAC S_TO_IN = (STRING[input_exp]bit:in) -> (flag,t_input): BIOP TRANSFORM_S. MAC IN_TO_S(INT n) = (t_input:in) -> (flag,STRING[in]bit): BIOP TRANSFORM_S.

MAC S_HUFF = (STRING[6]bit) -> (flag,t_huffman):BIOP TRANSFORM_US. MAC HUFF_S = (t_huffman) -> (flag,STRING[6]bit):BIOP TRANSFORM_US.

MAC BOOL_BIT = (boot:in) ->STRING[1] bit:

MAC BIT_BOOL= (bit:in)

ESAC.

OF b'11

ELSE f ESAC.

CASE in

OF tb1' ELSE b'0'

CASE in

```
result_range = 1 SL (result_exp-1), input_range = 1 SL (input_exp-1), max_octave=3, fino of octaves=max_octave +1, can not be less in this example#
                                                                                                                                                                                                                                                                                                                                                                                    #maximum shift value for quantisation constant#
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          ximage=319,#the xdimension -1 of the image, ie no of cols#
                                                                                                                                                                                                                                                                                                                                                     #length of 1D convolver input/output#
MAC BOOL_STRING(INT n) = ([n]bool:in) ->STRING[n] bit:
                                                                                                                           ELSE out[1] CONC BOOL_STRING(n-1)(n[2.n])
                                                                                                                                                                                                                        # defines the types used for the 2D wavelet chip#
                                                                                                                                                                                                                                                                                                                         Flength of result ariths
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           xsize = 10, #no of bits for ximage#
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 no_odave=max_odave+1, #"#
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              ysize = 9, #no of bits for yimage#
                           (LET out = BOOL_BIT in[1].
                                                                                                                                                                                                                                                                                                                    INT result exp=14,
                                                                                                                                                                                                                                                                                        #constant values#
                                                                                                                                                                                                                                                                                                                                                     input_exp=10,
                                                            OUTPUT IF n=1
                                                                                                                                                                                                                                                                                                                                                                                      qmax = 7,
                                                                                             THEN OUT
```

```
yimage=239 #the ydimension -1 of the image, le no of rows#
```

```
L_sparc_addr =NEW addr/(0..(1 SL max_octave)*( (ximage+1)*(yimage+1)+(bdmage+1))-1 ).
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 #un/down counter controll
TYPE ( result = NEW result( - (result range) .. (result range-1)),
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              cs = NEW(no select|select), #chip select control#
_updown= NEW(down|up), #up/down counter co
                                 | input = NEW input/ - (input_range)..(input_range-1))
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                diff= NEW(diff[nodiff], #diff or not in quantiser#
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 Ir/whar controls
                                                                                                                                                                                                                                                                                                                                   #address for result&dwt memory, le 1 frame#
                                                                                                                                                                                                                                                                                                                                                                                                   _octave=NEW oct/(0..(max_octave+1)),
                                                                                                                                                                                                                                                                                                     quant =NEW quant/(0..qmax)
                                                                                                                                                                                                                                                                                                                                                                                                                                                                    #bit string and boolean types types#
                                                                                                                                                                                                                                    row =NEW row/(0..yimage)
                                                                                                                                                                                                                                                                   carry =NEW carry/(0..1),
                                                                                                                                                                                                    col =NEW col/(0.xdmage)
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               load = NEW(write/read)
                                                                                                    lnp = NEW lnp/(0..1023), blk = NEW blk/(0..3),
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   1 reset = NEW(rst|no_rst)
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              Intra = NEW (intra inter)
                                                                  length= NEW len/(0..15),
                                                                                                                                                                  sub =NEW sub/(0..3),
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  bool = NEW (flt),
flag = NEW(error | ok),
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                bit = NEW b(0 | 11).
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    #control signals#
```

```
#convolver mux & and types#

t _mux = NEW(left|right),

t _mux = NEW(left|right),

t _mux4 = NEW(left|right),

t _add = NEW(add|subt),

t _direction=NEW(add|subt),

t _count control=NEW(count rst|count _carry),

t _count 2 = NEW(one|two),

#state types#

t _token = NEW (t_0|t_1|t_11|t_100|t_101),

t _mode= NEW(void|void|stit||stit||send||pf_stit||f_stop),
```

t_sparcport=(t_sparc_addr#wr_addr#,t_sparc_addr#rd_addr#,t_load#w/r#,t_cs#cs#) #generate random values for test memories#

channel factor= NEW (fuminance|color),

Hypes for the control of memory portsal

FN GEN_RANDOM_MEM = (bool:ck,t_reset:reset) ->t_input: BOOL_INT10 PRBS11(ck,reset) #These functions change types from boolean to inputeger and vice-# TYPE t_test = NEW(no)yes)

cycle = NEW(token_cycle|data_cycle|skip_cycle). state= NEW(start|up0|up1|zz0|zz1|zz2|zz3|down1),

decode = NEW (load_low|load_high)

huffman = NEW(pass|huffman)

high low = NEW(low/high),

t_fifo = NEW(nk_fifo|error_fifo), types for the octave control unit#

channel= NEW(y|u|v).

#versa. Supports 1 & 8 bit booleans.

1bit input to binary # FN INT_BOOL1=(!_input:k) ->bool:

CASE k OFInput/0:f, input/1:1

ESAC.

1 bit bool to input # FN BOOL_INT=(bool:b) ->t_input:

OFf:input/0, t:input/1 CASE

ESAC.

FN * = (1_input: a b) ->1_input: ARITH a*b.

FN % = (1 input:a b) ->1 input: ARITH a%b. FN - = (1 input:a b) ->1 input: ARITH a-b.

FN = =(I_input:a b) ->(_test: ARITH IF a=b THEN 2 ELSE 1 FI. FN + =(I_input:a b) ->(I_input: ARITH a+b.

FN CHANGE_SIGN = (I_input:l) ->1_input: #changes sign for 8-bit 2's# ARITH IF i<0 THEN 128+i #complement no, #

₩00

ELSEI

#gets sign for 2's# #complement nos # FN SIGN = (i input:i) ->bool: ARITH IF I<0 THEN 1

ELSE2

FN TEST_SIZE = (t_input:x) ->bool:
#tests to see if the input is bigger than an 8-bit inputeger#
ARITH IF ((x<=-128) AND (>>127)) THEN 1
ELSE 2 FI.

FN INTB_BOOL=(t_input:orig) ->[8]book:

SEQ VAR it:=input/0, #input variables# io:=CHANGE_SIGN(orig), b:=(f,f,f,f,f,SiGN(orig));

[INT n=1..7] (i1:=i0%inpud/2; b[n]:=INT_BOOL1(i0-inpud/2⁴1); i0:=i1); OUTPUT CASE TEST_SIZE orig #chec OFt: [8]?bool, #fit inputo an 8_bit value#

#checks to see if orig will#

ESAC

FN BOOL_INT8=([6]bool:b) ->1_input:#converts 6bit boolean to 2's#

VAR sum:=input/-128 * BOOL_INT(b[8]),

exp:=input/1;

ぜ

```
(BOOL_INT8(in1))+((input/258)*BOOL_INT8(in2))+((input/256)*BOOL_INT(in1[8])).
                                                                                                                                                                       FN BOOL_INT10=([10]bool:b) ->t_input:#converts 10bit boolean to 2's#
                                                                                                                                                                                                                                                                                                                                                                                                                                                 # convetrs a 16-bit no., (sbs.msbs) inputo inputeger form)#
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  #A 10 bit pats generator,feedback taps on regs 3 & 10.#
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              Phack because of sign extendi
                                                                                                                                                                                                                                          VAR sum:=input/-512 * BOOL_INT(0[10]).
                                                                                                                                                                                                                                                                                                       ( sum:=sum+exp*BOOL_INT(b[k]);
                                                                                                                                                                                                                                                                                                                                                                                                                                   FN BOOL_INT16 = (8) boot in 1 in 2) -> 1_input:
           sum:=sum+exp*BOOL_INT(b[k]);
                                                                                                                                                                                                                      #complement integer #
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 FN PRBS10 = (L_reset:reset) ->[10]bool:
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      #of Isb #
                                                                                                                                                                                                                                                                                                                             exp:=input/2 * exp
                                 exp:=input/2 * exp
                                                                                                                                                                                                                                                                     exp:=input/1;
                                                                                                                                                                                                                                                                                                                                                                           OUTPUT sum
                                                                                                                                                                                                                                                                                        [INT k=1.9]
                                                                              OUTPUT sum
[INT k=1..7]
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                BEGIN
                                                                                                                                                                                                                               SEO
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                №
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     8
                                                                                                                                                                                                           BEGIN
                                                                                                                                                                                                                                                                                                                                                                                                 ENO.
```

MAKE[10]MYLATCH:I, XNOR:xnor. FOR INT k=1..9 JOIN (reset, ifk) -> (ifk+1).

JOIN (reset, xnor) ->[[1], ([[10],[[3]] ->xnor.

OUTPUTI

№

#A 11 bit prbs generator, feedback taps on regs 2 & 11.# FN PRBS11 = (bool:ck,t_reset:reset) ->[10]bool:

MAKE[11]DFF{bool}:1, XOR:xor.

(ck,reset,[[k],f) ->[[k+1]. FOR INT k=1..10 JOIN

JOIN (ck,reset,NOTxor,f) ->[1]. (411],(2]) ->xor.

OUTPUT ((1..10)

FN PRBS16 = (bool:reset)->[16]bool:

#A 16 bit prbs generator, feedback taps on regs 1,3,12,16#

MAKE[16]MYLATCH:I,

XOR 4xor, NOT xnor. FOR INT k=1..15 JOIN (ck,reset,ljk).>\ljk+1].

JOIN (ck,reset,xnor) ->[1],
(I[1],I[3],I[16],I[12]) ->xor,
xor ->xnor.
OUTPUT (IINT k=1..16]IM)

FN PRBS12 = (clock:ck,bool:reset) ->[12]bool: #A 12 bit prbs generator,feedback taps on regs 1,4,6,12.#

MAKE[12]MYLATCH:I, XOR_4:xor, NOT:xnor. FOR INT k=1..11 JOIN (ck_reset,[k])->|[k+1].

JOIN (ck,reset,xnor) ->[1], (tl],![4],(6],[12]) ->xor, xor ->xnor.

OUTPUT (INT k=1..12)(k)

FN PRBS8 = (clock:ck,bootreset) ->[8]bool: #A 8 bit prbs generator,feedback taps on regs 2,3,4,8.#

MAKE [8]MYLATCH1,

XOR 4xor, NOT:xnor.

FOR INT k=1..7 JOIN (ck,reset,l[k]) ->l[k+1]. JOIN (ck,reset,xnor) ->[1], (I[2],I[3],I[4],I[8]) ->xor, xor ->xnor. OUTPUT ([INT k=1..8][k])

ENO.

#test for palmas chip#

TYPE 1_ini32 = NEW ini32/(-2147483000..2147483000).

FN RMS = (bool:ck,t_reset:reset,t_cycle:cycle,t_input:old new) ->t_int32: BEGIN

FN I 32 = (I_input:in) ->1 int32:ARITH in. FN DV = (I_int32:a b) ->1 int32:ARITH a%b. FN PL = (I_int32:a b) ->1 int32:ARITH a+b. FN MI = (I_int32:a b) ->1 int32:ARITH a-b. FN TI = (I_int32:a b) ->1 int32:ARITH a-b.

MAKEDFF_INIT(t_int32):old_enor.

LET err = [_32old Mi |_32new, err2 = (errTlerr) PL_old_error.

JOIN (ck,reset,CASE cycle OFdata_cycle:write

->old_error. ESAC, err2, int32/0) ELSE read

OUTPUT old_error

FN EQ = (1_input:a b) ->boot:ARITH IF a=b THEN 2 ELSE 1 FI.

FN SPARC_MEM = (t_input:in,t_sparc_addr:wr_addr,t_sparc_addr:rd_addr,t_load:rw_sparc#,t_cs:cs#)->t_input: RAM(input/0).

=(bool:ck,t_reset:reset,STRING(16]bit:buffer_in,t_direction:direction,t_load:fifo_read fifo_write) ->(STRING[16]bit,[2]t_fifo]: #fifo_full,empty# FN FIFO

FN FIFO_RAM = (STRING[16]bit:in,t_inp:w_addr rd_addr,t_load:rw_ffo) ->STRING[16]bit: RAM(b.000000000000000000).

FN FULL = (t_inp:in) ->t_ffio:ARITH IF in>1023 THEN 2 #ffio full#

FN INCR = (L_inp:in) ->L_inp:ARITH in+1.

FN EMPTY = (Linp:in) ->t fifo:ARITH IF in<0 THEN 2 #fifo empty#

ELSE 1

->t_inp:ARITH in-1. FN DECR = (t_inp:in)

MAKEDFF(t_inp):address,

FIFO_RAM:ram.

LET next = CASE direction
OFforward: CASE fifo_write
OFwrite:INCR address

ELSE address ESAC, inverse:CASE filo_read OFread:INCR address

ELSE address ESAC

ESAC.

JOIN (ck.reset,next,inp/0) ->addrccs, (buffer_in ,address,address,CASE direction

OF inverse:read, forward:fife_write

ESAC) ->ram.

OUTPUT (ram,(FUIL address, EMPTY address))

FN TEST_PALMAS = (bool:ck,t_reset:reset,t_direction:direction,t_intra:intra_inter,t_charmel_factor.channel_factor,t_input:q_int,t_quant:quant_norm,t_result:threshold comparison)

->(STRING[16]bit,#buffer_out#[2]t_toad#fifo_read fifo_write#,bool,bool,t_int32):

BEGIN

MAKE SPARC_MEM:new old_inv old_forw, FIFO:ffto, PALMAS:palmas_inv palmas_forw.

LET col_length = (IN_TO_S(9) input/31)[2],

row_length= (IN_TO_S(9) input(31)[2],

ximage_string = (IN_TO_S/9) input/32)[2],

yimage_string = (IN_TO_S(9) input/32)[2],

yimage_string_3 = (IN_TO_S(9) input/80)[2],

pro_forw = paimas_forw[1],

pro_inv = palmas_inv[1],

forw_frame_done = paimas_forw[7],

inv_frame_done = palmas_inv[7],

cycle = palmas_inv[8],

old_equal = CASE cycle
OF data_cycle:old_forw EQ palmas_inv[1]
ELSE t
ESAC.

NOS

```
(ck,reset,forward,intra_inter,channel_factor,q_int,quant_norm,b*000000000000000000000,new,old_forw, threshold,comparison,
                                                                                                                            #fifo[2][1],fifo[2][2]#ok_fifo,ok_fifo,col_length,row_length,ximage_string,yimage_string, yimage_string_3)
                                                                                                                                                                                                                     ->palmas forw,
#fix fifo full/empty logic later#
```

(ck,reset,inverse,intra_inter,channel_factor,q_int,quant_norm,fifo[1],new,old_inv, threshold,comparison, #fifo[2][1],fifo[2][2]#ok_fifo,ok_fifo,col_length,row_length,ximage_string,yimage_string,yimage_string_3] ->palmas_im,

#old forward mem, on forward use as normal, on inverse read values to compare with inverse# pro forw, CASE direction

OF forward:palmas_forw[2], inverse:palmas_inv[2]

ESAC, CASE direction

OF forward:palmas_forw[2], inverse:palmas_inv[2] ESAC, CASE direction

OF forward:palmas_forw[4][1], inverse:read

->old forw, ESAC) (palmas_inv[1],palmas_inv[2],palmas_inv[2],CASE direction inverse:palmas_inv[4][1] OF forward:read, ESAC) #(input/0.palmas_forw[2].palmas_forw[2],palmas_forw[3][1]) (input/0,CASE direction

->old_inv,

OF forward:palmas_forw[2], inverse:palmas_inv[2] ESAC, CASE direction

OF forward:palmas_forw[2],
inverse:palmas_inv[2]
ESAC,CASE direction
OF forward:palmas_forw[3][1],
inverse:read
ESAC) ->new,

OUTPUT (palmas_forw[5],palmas_forw[6],palmas_forw[7],old_equal,RMS(ck,reset,cycle,old_inv,new))

#test for palmas chip# TYPE t_int32 = NEW int32/(-2147483000..2147483000). FN RMS = (bool:ck,t_reset:reset,t_cyde:cyde,t_input:old new) ->t_int32: BEGIN

FN i 32 = (Lipput:in) ->t int32-ARITH in. FN DV = (Lint32:a b) ->t int32-ARITH a%b. FN PL = (Lint32:a b) ->t int32-ARITH a+b. FN MI = (Lint32:a b) ->t int32-ARITH a-b. FN TI = (Lint32:a b) ->t int32-ARITH a-b.

MAKE DFF_INIT(t_int32):old_error. LET err = I_32old MII_32new, err2 = (errTlerr) PL_old_error.

JOIN (ck,reset,CASE cycle
OF data_cycle.write
ELSE read
ESAC,err2,int320)

ESAC,err2,int32/0) ->old_error.

OUTPUT old_error END. EN FO = # input:a h) .>hoot-ARITI

FN EQ = (t_input:a b) ->boot:ARITH IF a=b THEN 2 ELSE 1 FI. FN SPARC_MEM = (t_input:in,t_sparc_addr:wr_addr,t_sparc_addr:rd_addr,t_load:rw_sparc#,t_cs:cs#)->t_input: RAM(input/0).

=(bool:ck,t_resel:reset,STRING[16]bit:buffer_in,t_direction:direction,t_load:fifo_react fifo_write) ->(STRING[16]bit,[2]t_fifo): #fifo_full,empty# FN FIFO_BIG BEGIN

FN FIFO_RAM = (STRING[16]bit:in,t_sparc_addr:wr_addr.rd_addr,t_load:rw_fifo) ->STRING[16]bit: HAM(b:000000000000000000)

FN FULL = (L_sparc_addr.in) ->t_ffo:ARITH IF in>1023 THEN 2 #ffo full#

FN INCR = (1_sparc_addr.in) ->1_sparc_addr.ARITH In+1.

FN EMPTY = (L. sparc_addrin) ->t_fifo:ARITH IF in<0 THEN 2 #fifo empty#

FN DECR = (t_sparc_addr.in) ->t_sparc_addr.ARITH in-1.

```
MAKE DFF(I sparc_addr):address,
FIFO_RAM:ram.
LET next = CASE direction
OF forward: CASE fifo_write
OF write:INCR address
ELSE address
ESAC,
inverse:CASE fifo_read
OF read:INCR address
ELSE address
```

ESAC

ESAC.

JOIN (ck,reset,next,addr/0) ->address,
(buffer_in,address,address,CASE direction
OF inverse:read,
forward.ffo_write
ESAC) ->ram.

FN TEST_PALMAS = (bodi:ck,t_resetreset, bookload_memory,t_direction:direction,t_intra:intra_inter, (ram,(FULL address,EMPTY address)) OUTPUT END.

FN 1EST_PALMAS = (bodi:dk,t_reset:reset, bookbad_memory,t_direction:direction,t_int t_channel_factor:channel_factor,[4]t_quant:quant_norm,[4]t_result:threshold, t_input: cof_length_in row_length_in ximage_string_in yimage_string_in, t_result:yimage_string_3_in)

->(bood#,t_int32#);

BEGIN

FN NEW_ADDRESS = (L_sparc_addr.in)

->t_sparc_addr. ARITH ((in +1) MOD 120000).

MAKE SPARC_MEM:new old_inv old_forw, FIFO_BIG:fffo, PRBS11:prbs,

DFF(t_sparc_addr):address, PALMAS:palmas.

col_length = (IN_TO_S(10) col_length_in)[2].

回

row_length= (IN_TO_S(9) row_length_in)[2],

ximage_string = (IN_TO_S{10} ximage_string_in)[2],

yimage_string = (IN_TO_S(9) yimage_string_in)[2],

yimage_string_3 = (I_TO_SC(11) yimage_string_3_in)[2],

pro= palmas[1],

random_data = BOOL_INT10 prbs,

frame_done = palmas[7],

cycle = palmas[8],

old_equal = CASE cycle

```
OF data_cycle:old_forw EQ palmas[1]
                ELSE
```

ESAC.

(ck, reset, direction, intra_inter, channel_factor, quant_norm, CASE direction #fix fifo full/empty logic later#

OFforward:b 00000000000000000

ELSE ffo[1]

ESAC, new, CASE direction

OF forward:old_forw

ELSE old_inv

#fifo[2][1],fifo[2][2]#ok_fifo,ok_fifo, col_length,row_length,ximage_string,yimage_string, yimage_string_3) ESAC, threshold

(ck.reset,(NEW_ADDRESS address), addr/0)

#old forward mem, on forward use as normal, on inverse read values to compare with inverses

->prbs,

OF t:DFF(Linput)(ck.reset.random_data,input/0) (CASE load memory

palmas(1) ELSE ESAC

, CASE load memory OF taddress

palmas[2], CASE load_memory OF twite palmas[2]

OF forward:palmas[4][1], CASE direction ELSE

SUBSTITUTE SHEET (RULE 26)

(ck,reset)

pakmas[2], CASE load_memory
OF twrite
ELSE CASE direction
OF forward:palmas[3][1], inverse:palmas[4][1 palmas[2] palmas[2], CASE load_memory OF twrite ELSE CASE direction OF forward:read, (CASE load_memory OF t:DFF(t_input)(ck,reset,random_data,input/0) inverse:read ESAC Inverseread ->new **ESAC** ESAC ESAC) palmas[1]
, CASE load memory
OF taddress ESAC) ESAC) , CASE load memory OF taddress palmas[2] (CASE load_memory ELSE ESAC, ELSE ESAC, ELSE ESAC

OUTPUT (old_equal#,RMS(ck,reset,cycle,old_inv,new)#)

#lest for palmas chip# TYPE t_int32 = NEW int32/(-2147483000..2147483000).

FN RMS = (bool:ck,t_reset:reset,t_cyde:cyde,t_input:oid new) ->t_int32: BEGIN

FN I 32 = (I_input:in) ->t_int32:ARITH in. FN DV = (I_int32:a b) ->t_int32:ARITH a%b. FN PL = (I_int32:a b) ->t_int32:ARITH a+b. FN MI = (I_int32:a b) ->t_int32:ARITH a-b. FN TI = (I_int32:a b) ->t_int32:ARITH a-b.

MAKEDFF_INIT{_int32}:old_error.

LET err = 1_320kd MI1_32new, err2 = (errTlerr) PL_old_error. JOIN (ck,reset,CASE cycle OFdata_cycle:write ELSE read ESAC,err2,int32/0) ->old_error.

OUTPUT old_error

FN EQ = (Linput:a b) ->boot:ARITH IF a=b THEN 2

FN SPARC_MEM = (t_input:in,t_sparc_addr:wr_addr,t_sparc_addr:rd_addr,t_load:rw_sparc#,t_cs:cs#)->_input: RAM(input/0).

FN FIFO = (bool:ck,1_reset:reset,STRING[16]bit:buffer_in,1_direction:direction,1_load:fifo_read fifo_write) ->(STRING[16]bit,[2]t_fifo): #fito_full,empty#

BEGIN

FN FIFO_RAM = (STRING|16|bit:in,t_inp:wr_addr.rd_addr,t_load:rw_ffo) ->STRING[16]bit:

RAM(b*0000000000000000000).

FN FULL = (t_inp:in) ->t_fifo:ARITH IF in>1023 THEN 2 #fifo full# ELSE 1

FN INCR = (L_inp:in) ->t_inp:ARITH in+1.

FN EMPTY = (L_inp:in) ->t_fffo:ARITH IF in<0 THEN 2 #fffo empty# ELSE 1 F1.

FN DECR = (Linp:in) ->Linp:ARITH in-1.

MAKEDFF(t_inp):address, FIFO_RAM:ram.

LET ned = CASE direction
OFforward: CASE (flo_write
OFwrite:INCR address
ELSE address
ESAC,

inverse.CASE fife_read OFread:INCR address ELSE_address

ESAC.

JOIN (ck,reset,next,Inp/0) ->address,
(buffer_in_address,address,CASE direction
OF_inverse:read,
forward:fifo_write
ESAC) ->ram.

OUTPUT (ram, (FULL address, EMPTY address))

9

FN TEST_PALMAS = (bool:ck,t_reset:reset, bool:load_memory,t_direction:direction,t_intra:intra_inter,t_channel_tactorxhannel_tactor, t input:q int,t_quant:quant_norm,t_result:threshold comparison)

->(bood,t_int32):

BEGIN

FN NEW_ADDRESS = (L_sparc_addr:In) ->t_sparc_addr: ARITH ((in +1) MOD 120000).

MAKE SPARC_MEM:new old_inv old_forw, FIFO:fifo, PRBS11:prbs,

DFF(\(\text{sparc_addr}\):address, PALMAS:palmas. LET col_length = (IN_TO_S(10) input/31)[2],

row_length= (IN_TO_S(9) input/31)[2],

ximage_string = (IN_TO_S(10) input/32)[2],

yimage_string = (IN_TO_S(9) input/32)[2],

yimage_string_3 = (I_TO_SC(11) result/80)[2],

pro= palmas[1],

random_data = BOOL_INT10 prbs,

frame_done = palmas[7],

cycle = palmas[8],

old_equal = CASE cycle
OF data_cycle:old_forw EQ palmas[1]
ELSE1
ESAC.

#fix fifo full/en.pty logic later#

(ck, reset, direction, intra_inter, channel_factor, q_int, quant_norm, filo[1], new, CASE direction

OF forward:old forw

ELSE old_inv _ ESAC, threshold,comparison,

#fifo[2][1],fifo[2][2]#ok_fifo,ok_fifo,col_length,row_length,ximage_string,yimage_string, yimage_string_3)

(ck,reset,(NEV/_ADDRESS address), addr/0)

-> address,

(ck,reset)

#old forward mem, on forward use as normal, on inverse read values to compare with inverse.

(CASE load_memory

OFt:DFF(Linput)(ck,reset,random_data,input/0)

ELSE palmas[1]

ESAC, CASE load memory

ELSE palmas[2] OF traddress

palmas[2], CASE load memory OF twitte ESAC,

OF forward:palmas[4][1], ELSE CASE direction inverse:read

ESAC ESAC)

CASE load memory

OF t:DFF(t_input)(ck_reset,random_data,input/0)

palmas[1]

```
palmas[2], CASE load_memory
OF twrite
ELSE CASE direction
OF forward:palmas[3][1],
                                                                      OF forward:palmas[4][1]
                   palmas[2]
palmas[2], CASE load_memory
OF twrite
ELSE CASE direction
                                                                                                          ->old_inv,
                                                                                   inverse:read
                                                                                                                                                                                                                                                   inverse:read
                                                                                                                                                                                                                                                                           ->new
                                                                                             ESAC
ESAC , CASE load memory OF taddress
                                                                                                        ESAC)
                                                                                                                                                              , CASE load memory
OF taddress
                                                                                                                                                                                                                                                                        ESAC)
                                                                                                                                                                                      palmas[2]
                                                                                                                                       OF t:random_data
ELSE input/0
                                                                                                                             (CASE load_memory
                       ELSE
ESAC,
                                                                                                                                                                                     ELSE
ESAC,
                                                                                                                                                              ESAC
```

(ck,reset,CASE direction OF inverse:b*00000000000000*, forward:palmas[5]

direction, palmas [6][1], palmas [6][2])

ESAC

OUTPUT (old_equal,RMS(ck,reset,cycle,old_inv,new)) END.

SUBSTITUTE SHEET (RULE 26)

- 644 -

APPENDIX C

```
7/24/93 3:J9 PM
                       Engineering:KlicsCode:CompPics:Top.a
.....
   © Copyright 1993 KLICS Limited
   All rights reserved.
   Written by: Adrian Lewis
   630X0 Fast Top Octave
       seg
                  'klics'
       macro
       TOPX
                 &DG, &HG, &old, &XX
       swap
                  &HG
                                          : HG=G1H0
       move.w
                  &DG,&XX
                                         ; XX=G0
       neg.w
                                         ; DG=D(-G0)
                  &DG
       add.w
                                         ; DG=DD
                  &HG,&DG
       add.w
                  £XX, £HG
                                          ; HG=G1D
       swap
                  £HG
                                          ; HG=DG1
       move.1
                  £DG, &old
                                         ; save DD
       endm
       macro
       TOPY
                  &HGO, &newO, &HG1, &new1, &XX
       move.1
               &new0,&XX
&new1,&HG1
                                         ; read HG
       move.l
                                         ; read HG
                 &HG1, &HG0
       move.1
                                         ; copy HG
       add.l
                 &XX.&HG1
                                         ; newl=H1G1
       sub.1
                  &XX, &HG0
                                         ; new0=H0G0
       endm
       macro
                &DGO, &HGO, &newO, &oldO, &DG1, &HG1, &new1, &old1, &XX
                  &HG0, &new0, &HG1, &new1, &XX
       TOPY
       TOPX
                  &DGO, &HGO, &oldO, &XX
       TOPX
                  &DG1, &HG1, &old1, &XX
       endm
       macro
                  &DG, &HG, &new, &old, &XX
       move.l
                  &new,&HG
       TOPX
                  &DG, &HG, &old, &XX
       macro
       TOPE
                 &DG, &old, &XX
                  &DG, &XX
                                         ; XX=DG
       move.l
       swap
                  £XX
                                         ; XX=GD
                                        : DG=DD
                  &XX, &DG
       move.w
                  &DG, &old
                                         ; save DD
       move.l
```

- 646 -

Engineering:KlicsCode:CompPict:Top.a

	endm	-	
TopBwd	FUNC E	EXPORT	
PS STC	RECORD DS.L	9 1	
dst	DS.L	1	
width	DS.L	2	
neight	DS.L	1.	
•	ENDR		
	link	a6. #0	; no local variables
	movem.l	d4-d7/a3-a5,-(a7)	; store registers
•			
	movea.l	PS.src(a6),a0	; read src
	move.l	PS.height(a6),d7	; read height
•	move.l move.l	PS.width(a6),d6 a0.a1	; read width
	move.l	PS.dst(a6),al	; read dst
	_		
	move.i	d6,d5	; inc = width
	add.1 move.1	d5,d5 d5,a4	; inc*=2 ; save inc
	move. I	٠, ۵٠, ۵٠	; save me
	lsr.1	*1,d7	; height/=2
	subq.l	#2,d7	; height-=2
	lsr.1	#2,d6	; width/=4
	subq.1	#2.d6	; width==2
		36 48	and the second second
	move.l move.l	d6,d5 (a0)+,d0	; ccount=width ; d0=*new0++
			, qu- newov+
@do1	TOPH	d0.d1.(a0)+.(a1)+.d4	
	TOPH dbf	d1,d0,(a0)+,(a1)+,d4 d5,9do1	
	TOPH	d0,d1,(a0)+,(a1)+,d4	; while -1!=ccount
	TOPE	dl,(al)+,d4	
9403	1	20.22	
edo2	move.l move.l	a0,a2 a1,a3	<pre>; new0=new1 ; old0=old1</pre>
	adda.1	a4, a0	: newl+=inc
	adda.l	a4.al	; old1+=inc
•	move.l	d6, d5	; ccount=width
	TOPY	d2,(a2)+,d0,(a0)+,d4	
@do3	TOPBLOCK	d2,d3,(a2)+,(a3)+,d0,d1	(a0)+,(a1)+,d4
	TOPBLOCK	d3.d2,(a2)+,(a3)+,d1.d0.	
	dbf	d5.9do3	; while -1!=ccount
	TOPBLOCK	d2,d3,(a2)+,(a3)+,d0,d1,	(a0)+, (a1)+,d4
	TOPE	d1,(a1)+,d4	,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
	TOPE	d3,(a3)+,d4	
	dbf	d7,0do2	; while -1!=height
	move.1	d6,d5	; ccount=width
	add.l	•1,d5	; d0=*new0++
edo4	move.l	(a3)+, (a1)+	; copy prev line
· •	move.1	(a3)+,(a1)+	
	dbf	d5.8do4	; while -1!=ccount
	movem.1	(a7)+,d4-d7/a3-a5	; restore registers

- 647 -

Engineering:KlicsCode:CompPict:Top.a

unik rts

; remove locals
; return

ENDFUNC

ENC

Engineering: KlicsCode: CompPict: Table.a

```
© Copyright 1993 KLICS Ltd.
    All rights reserved.
    680X0 Table Lookup RGB/YUV code
        machine
                      MC68030
                      'klics'
        seg
        if &TYPE('seg') = 'UNDEFINED' then
        seg
                      &seg
        endif
MKTABLE FUNC
                 EXPORT
PS
        RECORD
                      8
Table
        DS.L
                      1
        ENDR
        link
                      a6,#0
        movem.1
                      d4-d7/a3-a5,-(a7)
                                                 ; store registers
        move.1
                      PS.Table(a6),a0
                                                 :Table is (long)(2U+512) (long)(512-(6'
        clr.1
                      90
                                                 ;U value
@MakeLoop
                      #512,d1
        move.w
                                                 ;512
        move.l
                      d0,d2
                                                 ; U
        move.w
                      d2,d3
                                                 ; U
        add.w
                      d2,d2
                                                 : 20
                                                ;20 + 512--
        add.w
                      d1,d2
                      #2,d2
        lsr.w
                      d2, (a0)+
d2, (a0)+
                                                :Place 1st word
        move.w
        move.w
                                                ;Place 2nd word
        add.w
                      d3,d3
                                                ; 2U
        move.w
                      d3,d2
                                                ; 2U
        add.w
                      tb, tb
                                               - ; 4U
        add.w
                      d2.d3
                                                ; 6U
        asr.w
                      #4.d3
                                                :60/16
        sub.w
                      d3,d1
                                                :512 - (60/16)
        lsr.w
                      #2.dl
                      d1,(a0)+
d1,(a0)+
                                                ;Place 1st word ;Place 2nd word
        move.w
        move.w
        add.w
                      #1.d0
                      #$0200,d0
        CTTP.W
        bne
                      @MakeLoop
        move.1
                      #$00000200.d0
                                                ;U value
        clr.1
@MakeNegLoop
                      #512.dl
                                                :512
        move.w
        move.w
                      d0.d2
                                                ; U
```

Table

DS.L

1

Engineering:KlicsCode:CompPict:Table.a

```
or.w
                     ₹$FC00,d2
        nove. w
                     d2.d3
                                                ; ប
        add. w
                     d2.d2
                                                : 20
        add.w
                     d1,d2
                                                :2U + 512
        agr.w
                     #2.d2
        move.w
                     d2, (a0)+
                                               :Place 1st word
        move.w
                     32, (a0)+
                                               ;Place 2nd word
                                               ; 2U
        add.w
                     d3.d3
                                               ; 2U
        move.w
                     d3.d2
        add.w
                     d3.d3
                                               : 4U
        add.w
                     d2,d3
                                               ; 6U
        asr.w
                     #4,d3
                                               ;60/16
        sub.w
                     d3.d1
                                               :512 - (60/16)
        asr.w
                     $2.dl
        move.w
                     d1, (a0)+
                                               ; Place 1st word
        move.w
                     d1,(a0)+
                                               :Place 2nd word
        add.1
                     01.d0
        add.I
                     #1,d4
        CTUD.W
                     #$0200.d4
        bne
                     @MakeNegLoop
        movem.1
                     (a7)+.d4-d7/a3-a5
                                               ; restore registers
        unlk
                     a6
                                               ; remove locals
        rts
                                               ; return
        ENDPUNC
        macro
        FIXOV
                     &V, &SP1, &SP2
        move.w
                     &V.&SP1
        clr.b
                     &SP1
        andi.w
                     #$3PFF, &SP1
        sne
                     &SP1
                     #13,4SP1
4SP2
        best
        seq
        or.b
                     &SP1.&V
                     &SP2,&V
        swap
                     ٤V
                     &V, &SP1
        move.w
        clr.b
                     &SP1
        andi.w
                     #$3FFF, &SP1
        sne
                     4SP1
        best
                     #13,&SP1
        seq
                     &SP2
        or.b
                     &SP1,&V
        and.w
                     &SP2, &V
        swap
        endm
        if &TYPE('seg')≠'UNDEFINED' then
        seg
                     &seg
        endi f
YUV2RGB4
            FUNC
                     EXPORT
PS
        RECORD
```

Pdo

; uv2rgb(*!/++, *V++)

```
Engineering: KlicsCode: CompPict: Table.a
 pixmap DS.L
                       1
         DS.L
                       1
 •
          DS.L
                       1
          DS.L
                       1
 area
          DS.L
 width
         DS.L
                       1
 cols
         DS.L
                       1
         ENDR
 ٦S
         RECORD
                       0, DECR
 inc
         DS.L
 width
         DS.L
 fend
         DS.L
                       1
 count
         DS.L
 LSize
         EOU
         ENDR
*void YUVtoRGB(Ptr TablePtr.long *pixmap.short *Yc.short *Uc.short *Vc.long area.l
 ·long
              inc. lwidth. fend, count;
         a0 - Y0, a1 - Y1, a2 - U, a3 - V, a4 - pm0, a5 - pm1 d0..6 - used, d7 - count
:
         link
                      a6. #LS.LSize
                                                ; save locals
         movem.l
                      d0-d7/a0-a5,-(a7)
                                                ; store registers
         move.1
                      PS.pixmap(a6),a4
                                                : pm0=pixmap
         move.l
                      a4,a5
                                                : pml=pm0
         move.1
                      PS.Y(a6),a0
                                                  Y0=YC
         move.1
                      a0,a1
                                                ; Y1=Y0
         move.1
                      PS.U(a6),a2
                                                ; U=Uc
         move.1
                      PS.V(a6),a3
                                                ; V=Vc
         move.1
                      PS.area(a6),d7
                                                ; fend=area
         1s1.1
                      #2,d7
                                                ; fend<<=2
         add.l
                      a4,d7
                                                ; fend+=pm0
         move.l
                      d7, LS. fend(a6)
                                                ; save fend
                      PS.width(a6),d5
         move.l
                                                ; width=width
         move.1
                      d5.d7
                                                ; count=width
         asr.l
                      #1,d7
                                                ; count>>=1
         subq.1
                      #1,d7
                                                ; count-=1
         move.l
                      d7, PS. width(a6)
                                                ; save width
         add.1
                      d5, d5
                                                : width*=2
         add.1
                      d5, a1
                                               : Y1+=width
        add.l
                     d5,d5
                                                ; width = 2
        move.1
                     d5, LS. width(a6)
                                               ; save width
        move.l
                     PS.cols(a6),d4
                                               ; inc=cols
        1s1.1
                     #2,d4
                                                ; inc<<=2
        add.1
                     d4, a5
                                               ; pml+=inc
        add.1
                     d4.d4
                                               ; cols*=2
        sub.1
                     d5.d4
                                               ; inc now 2*cols-width bytes
        move.1
                     d4.LS.inc(a6)
                                               ; save inc
        move.1
                     a6.-(sp)
        move.l
                     PS. Table (a6), a6
; Colors wanted are:
    RED
            = (Y + 2V + 512) / 4
= (Y - V + 512 - (6U/16)) / 4
                                                       UTable part is for (2V + 512)
    GREEN
                                                       UTable part is for (512 - (60
    BLUE
            = (Y + 2U + 512) / 4
```

UTable part is for (2U + 512)

Engineering:KlicsCode:CompPict:Table.a

```
d1 - ra= d2 - ga, d3 - ba,
                                               d4 - rb. d5 - gb/512, d6 - bb
          move. w
                         (a2)+,d2
                                                       ; U
          bea
                         @DoQuickU
          and.w
                         #$03FF.d2
          move.l
                         (a6,d2.w.8),d3
                                                       :BLUE.Get (2U + 512)/4 for Blue = (Y +
          move.l
                                                       Dup for second pair GREEN, Get (512 - (6U/16))/4 for Gree.
                         d3.d6
          move.1
                         4(a6,d2.w*8),d5
@DidQuickU
          move.w
                         (a3)+,d1
          beσ
                         @DoQuickV
                                                       ; if zero then handle using the quick m
          move.w
                         d1,d4
          ASI.W
                         #2,d1
          sub.w
                         d1,d5
                                                       ;GREEN. Get (512 - (6U/16) - V)/4 for .
          move.w
                         d5,d2
          GAWR
                         d5
          move.w
                         d2.d5
          move.l
                         d5.d2
                                                       ;Dup for second pair
          and.w
                         $$03FF,d4
         move.1
                         (a6,d4.w*8),d4
                                                       :RED, Get (2V + 512)/4 for Red = (Y +
         move.1
                         d4,d1
          bra
                         @TestEnd
@DoQuickU
         move.1
                         #$00800080,d3
                                                      ;BLUE.Get (2U + 512)/4 for Blue = (Y +
         move.1
                                                      Dup for second pair GREEN, Get (512 - (6U/16))/4 for Green
                         d3,d6
         move.1
                         d3.d5
         bra
                         @DidQuickU
@DoOuickV
                                                      ;GREEN, Get (512 - (60/16) - V)/4 for ;RED, Get (2V + 512)/4 for Red = (Y +
         move.1
                         d5.d2
         move.1
                         #$00800080,d4
         move.1
                         d4.d1
                                                      ; Dup for second pair
@TestEnd
          ; add Ya to RGB values - FETCHY (a0)+,d0,d1,d2,d3
                         (a0)+,d0
         move.1
         AST.W
                         #2,d0
         swap
                         d٥
                         #2,d0
         asr.w
                                                      ;Y is -128 to +127
;RED, Get (Y+ 2V + 512) for Red = (Y +
;GREEN, Get (Y + (512 - (6U/16)) - V)
;BLUE,Get (Y + (2U + 512) for Blue = (
          SWAD
                        d0
         add.l
                        d0.d1
         add.1
                        d0,d2
         add.1
                        d0.d3
         ; add Yb to RGB values - FETCHY2 (al)+,d0,d4.d5,d6
                        (a1)+,d0
         move.l
         asr.w
                        #2,d0
                        đ0 .
         swap
                        #2,d0
         asr.w
                                                      ;Y is
         swap
                        d0
                                                                    -128 to +127
                                                      ;RED, Get (Y+ 2V + 512) for Red = (Y + :GREEN, Get (Y + (512 - (6U/16)) - V) ;BLUE,Get (Y + (2U + 512) for Blue = (
         add.1
                        d0,d4
         add.1
                        d0.d5
         add.1
                        d0, d6
         move.1
                        d1.d0
         or.1
                        d4, d0
         or.1
                        d2,d0
                        d3,d0
         or.l
         or.l
```

```
Engineering: KlicsCode: CompFict: Table.a
         or.1

→ d6.d0

         and.1
                      *SFF00FF00.d0
         bre
                      fover
                                               : if overflow
3ok
         ; save RGBa - MKRGB d1,d2,d3,(a4)+
         lsl.1
                      #8.d2
                                               : G=G0GC (12)
         or.l
                      d3.d2
                                               : G=GBGB (12)
                      d1.d3
         move.1
                                               : B=0R0R (12)
         swap
                                              : B=OROR (21)
                      d3
         move.w
                      d2.d3
                                               : B=0RGB (2)
         swap
                      aг
                                               ; G=GBGB (21)
         move.w
                      d2.d1
                                               ; R=0RGB (1)
         move.l
                      d1,(a4)+
                                               : *RGB++=rgb (1)
        move.l
                                               : *RGB++=rgb (2)
                      d3.(a4)+
         ; save RGBb - MKRGB d4.d5.d6.(a5)+
                     #8.d5
        lsl.l
                                               : G=G0G0 (12)
                     d6,d5
         or.l
                                               : G=GBGB (12)
        move.1
                     d4,d6
                                               ; B=0R0R (12)
         swap
                      d6
                                               : B=0ROR (21)
         move.w
                      d5, d6
                                               : B=0RGB (2)
         swap
                     d5
                                              ; G=GBGB (21)
        move.w
                     d5.d4
                                              : R=0RGB (1)
        move.1
                     d4.(a5)+
                                              : *RGB++=rgb (1)
         move.l
                     d6, (a5)+
                                              : *RGB++=rgb (2)
        dbf
                     d7, 3do
                                              ; while
        move.1
                     (sp)+, a6
        adda.1
                     LS.inc(a6),a4
                                              : pm0+=inc
        adda.1
                     LS.inc(a6),a5
                                              : pml+=inc
        adda.1
                     LS.width(a6), a0
                                              ; Y0+=width
        exg.l
                     a0,a1
                                              ; Y1<->Y0
        move. 1
                     PS.width(a6),d7
                                              : counc = width
                     LS.fend(a6),a4
        cmpa.1
                                              : pm0<fend
        blc.w
                     @do2
                                              ; while
        movem.1
                     (a7)+.d0-d7/a0-a5
                                              ; restore registers
        unlk
                     a6
                                              ; remove locals
        TE S
@do2
        move.l
                     a6. - (sp)
        move.1
                     PS.Table(a6),a6
        bra
                                              ; return
OFixIt
                     #31,d0
        btst
                                              ;See if upper word went negative
        beq and.1
                     @DITopNotNeg
                     #50000FFFF, d0
                                              :Pin at zero
@D1TopNotNeg
        btst
                     #24,d0
                                             ; See if upper word went too positive
        beq
                    QD1TopNotPos
                     #$0000FFFF, do
        and.1
                                              : Mask old data out
        or.l
                     #$00FF0000,d0
                                             :New data is maxed
@DlTopNot Pos
        btst
                    #15.d0
                                             :See if lower word went negative
        beg
                    @D1BotNotNeg
```

Engineering:KlicsCode:CompPict:Table.a

```
*$FFFF0000, d0
         and. 1
                                                  :Pin at zero
@DlBotNotNeg
         best
                      #8.d0
                                                  :See if lower word went too positive
                      @D1BotNotPos
         pec
                      SFPFF0000.d0
         and.1
                                                  : Mask old data out
        or.l
                      $5000000FF, do
                                                  :New data is maxed
@DiBotNot Pos
        rts
@over
        {\tt move.l}
                      d1.d0
                      @FixIt
        bsr
        move.l
                      d0,d1
        move.l
                      d2,d0
        bsr
                      @FixIt
        move.l
                      d0.d2
        move.1
                      d3,d0
        bsr
                      OFIXIt
                      ۵0, طع
        move.1
                     d4.d0
@PixIt
d0.d4
        move.l
        bsr
        move.1
        move.1
                     d5.d0
        bsr
                     OFIXIE
        move.1
                     d0.d5
                     d6.d0
@FixIt
d0.d6
        move.l
        bsr
        move.l
        bra
                     Øok
        ENDFUNC
       END
```

1

Engineering: KlicsCode: CompPict: KlicsUtil.a

```
© Copyright 1993 KLICS Limited
    All rights reserved.
    Written by: Adrian Lewis
    68000 Klics Utilities
                 'klics'
        seg
KLCopy FUNC
                 EXPORT
    KLCOPY(shor: 'src, short 'dst, int area);
PS
         RECORD
                      8
src
         DS.L
                      1
dst
         DS.L
                      1
end
         DS.L
                      1
         ENDR
         link
                      a6.#0
                                                : no local variables
         move.l
                      PS.src(a6),a0
                                                ; short *src
         move.1
                      PS.dst(a6),al
                                                ; short *ast
         move.l
                      PS.end(a6),d3
                                                ; long area
         lsr.l
                      #4.d3
                                                ; in words(x8)
         subq.1
                      #1.d3
                                                ; area-=1
3do
         move.1
                      (a0)+,(a1)+
                                                ; *dst++=*src++
; *dst++=*src++
         move.1
                      (a0) +, (a1) +
         move.l
                      (a0)+,(a1)+
                                                ; *dst++=*src++
                      (a0)+, (a1)+
         move.1
                                                : *dst++=*src++
                      (a0)+,(a1)+
         move.1
                                                : *dst++=*src++
         move.1
                      (a0)+,(a1)+
                                                : *dst++=*src++
         move.1
                      (a0)+,(a1)+
                                                : *dst++=*src++
         move: 1
                      (a0)+,(a1)+
                                                : *dst++=*src++
         dbf
                      d3,edo
                                                ; if -1!=--area goto do
         unlk
                                                : remove locals
         rts
                                                ; return
         ENDFUNC
KLHalf FUNC
               EXPORT
    KLHALF(short *src, short *dst, long width, long height);
Dimensions of dst (width, height) are half that of src
PS
        RECORD
                      8
SEC
        DS.L
                      1
dst
        DS.L
                      1
width
        DS.L
height
        DS.L
                     1
        ENDR
        link
                     a6.#0
                                                ; no local variables
        movem.1
                                              ; store registers
                     d4, -(a7)
        move.l
                     PS.src(a6), a0
                                              ; short *src
        move.1
                     PS.dst/a61.al
                                               : short *dst
```

Engineering: KlicsCode: CompPict: KlicsUtil.a

```
_PS.width(a6),d2
        move.l
                                                : long width
                     PS.height(a6).d3
        move.1
                                                : long height
        subq. 1
                     #1.d3
                                                : height-=1
6qo_7
        move. l
                     d2.d4
                                                ; count=width
        lsr.l
                      42.d4
                                                : count /= 2
        subq. 1
                      #1.d4
                                                : count-=1
3do_x
                      (a0)+.d0
        move. 1
                                                ; d0=*src++
        nove.w
                     (a0)+.d0
                                                : d2="src++
        addq.1
                     #2.a0
                                                : src+=l short
        move. :
                     d0.(a1)+
                                               ; *dst++=d0
                     (a0) + , d0
        move. 1
                                               ; d0=*src++
        move.w
                      (a0) + .d0
                                               ; d2=*src++
        addq.1
                     #2, a0
                                               : src+=1 short
                     d0, (a1)+
        move.l
                                                : *dst++=d0
        dbf
                     d4.9do_x
                                               ; if -l!=--width goto do_x
        adda.l
                                               ; skip a quarter row
                     d2, a0
                                               ; skip a quarter row
        adda.1
                     d2, a0
        adda.l
                     d2.a0
                                               ; skip a quarter row
                     d2, a0
        adda.l
                                               ; skip a quarter row
        dbf
                     d3,0do_y
                                               ; if -1!=--height goto do_v
        movem.1
                     (a7)+,d4
                                               ; restore registers
        unlk
                     a6
                                               ; remove locals
        ITS
                                               ; return
        ENDFUNC
KLZero FUNC
                 EXPORT
    KLZERO(short *data, int area);
PS
        RECORD
                     8
data
        DS.L
                     1
end
        DS.L
                     1
        ENDR
        link
                     a6.90
                                               ; no local variables
        move.1
                     PS.data(a6),a0
                                               : short *data
        move.l
                     PS.end(a6),d3
                                                 long area
                     #3.d3
#1.d3
        lsr.l
                                               ; in words(x4)
        subg. 1
                                               ; area-=1
@do
        clr.1
                     (a0) +
                                               : *dst++=*src++
        clr.l
                     (a0) +
                                               : 'dst++='src++
                     (a0)+
                                               'dst++='src++
        clr.1
        clr.1
                     (a0) +
                                                'dst++='src++
                     d3. @do
        dbf
                                               ; if -l!=--area goto do
        unlk
                                               ; remove locals
        rts
                                               ; return
      · ENDFUNC
CLEARA2 FUNC
                EXPORT
                     #0,a2
        rts
        END
```

Engineering: KlicsCode: CompPict: KlicsEncode.h

```
9 Copyright 1993 KLICS Limited
      All rights reserved.
     Written by: Adrian Lewis
typedef struct (
                                  /* User - Bytes per frame in input stream */
/* User - Bytes per frame in output stream */
/* User - Buffer size (bytes) */
      int
                 bpf_in,
                  bpf_out,
                  buf_size:
      Boolean intra.
                                    /* Calc - Compression mode intra/inter */
                                    /* User - Automatic quantization for rate control */
/* User - Theoretical buffer on/off */
                  auto_q,
                  buf_sw:
                                   /* User - Starting quantiser value */
/* User - Threshold factor */
/* User - Comparison factor */
/* User - Octave weighting factors */
      float
                  quant,
                  thresh.
                  compare,
                  base(5);
                 buffer,    /* Calc - Current buffer fullness (bytes) */
prevbytes,    /* Calc - Bytes sent last frame */
prevquact:    /* Calc - Quantisation/activity for last frame */
      int
      double tmp_quant: /* Calc - Current quantiser value quant */
) KlicsEDataRec:
typedef struct (
      KlicsSeqHeader
                                    segh;
      KlicsFrameHeader
                                    frmh;
      KlicsEDataRec
                                    encd;
      Buffer
                                    buf;
) KlicsERec, *RlicsE;
```

```
·----
    © Copyright 1993 KLICS Limited
    All rights reserved.
    Written by: Adrian Lewis
 680x0 KlicsDecode code
    Fast code for:
        3/2 octave input stream
        2/1 octave output image
                    'klics'
        seg
        include
                   'Bits3.a'
        include
                    'Traps.a'
       machine MC68030
..........
    Data stream readers:
    XDELTA, XVALUE, SKIPHUFF, XINT
        macro
        XDELTA
                   &addr, &step, &ptr, &data, &bno, &spare
        buf_rinc
                   éptr.&data,&bno
        buf_get
                   £data, £bno
        beq.s
                                          ; if zero write ; set up count
                   equit
        moveq
                   #6, £spare
        buf_get
                   Edata, Ebno
                                          ; read sign
                  doneg
       bne.s
                                          ; if negative -> doneg
@dopos buf_get
                   &data, &bno
        dbne
                   &spare, @dopos
                                          ; if --spare!=-1
                   Ofndpos
       bne.s
       move.1
                   &data,&spare
                                          : spare=data
       subq.b
                   #7.Ebno
                                          ; bno-=6
       ler.1
                                          ; spare>>=bno
; spare AND= mask
                   &bno.&spare
       andi.w
                   #$007F.&spare
       add.w
                   #8, &spare
                                          : spare+=9
                   Owrite
       bra.s
@fndpos neg.w
                   &spare
                                          ; bits-=bits
       addq.1
                   #7,&spare
                                          : bits+=8
       bra.s
                   GWILLE
@doneg buf_get
                   &data,&bno
                   &spare, @doneg
@fndneg
       dbne
                                         ; if --spare!=-1
       bne.s
       move.1
                   &data,&spare
                                         ; spare=data
; bno-=6
       d.pduz
                   #7, abno
       lsr.1
                   &bno, £spare
                                         ; spare>>=bno
: spare AND= mask
       andi.w
                   #S007F.&spare
```

- 658 -

```
Engineering:KlicsCode:CompPict:KlicsDec2.a
```

```
add.w =
                     48.&spare
                                            : spare+=9
        neg.w
                     &spare
        bra.s
                     Gwrite
?fndneg subq.l
                     *7.&spare
                                               : level-=8
Awrite Isl.w
                     &step.&spare
                                               : level<<=step
        swap
                     &step
        add.w
                     &step, &spare
        swap
                     &step
        add.w
                     &spare.&addr
                                               : *addr=delta
@quit
        endm
        macro
        XVALO
                     &addr. &scep, &ptr. &data. &bno. &spare
        clr.w
                     &spare
        buf_rinc
                     &ptr.&data.&bno
        buf_get
                     &data, &bno
                                               ; if zero write
        beq.s
                     equit
                                              ; set up count
        moveq
                     #6, &spare
        buf_get
                     &data, &bno.
                                               : read sign
        bne.s
                     @doneg
                                               ; if negative -> doneg
dopos buf_get
                     &data, &bno
        dbne
                     &spare, @dopos
                                              ; if --spare! =-1
        bne.s
                     efndpos
        move.1
                     &data, &spare
                                              : spare=data
        subq.b
                     #7,4bno
                                              ; bno-=6
                     £bno.£spare
#$007F.£spare
        lsr.l
                                              ; spare>>=bno
        andi.w
                                              : spare AND: mask
        add.w
                     #8.&spare
                                              ; spare+=9
        bra.s
                     Owrite
@fndpos neg.w
                     & spare
                                              ; bits-=bits
        addq.1
                     $7,&spare
                                              ; bits+=8
        bra.s
                     Swrite
3doneg buf_get
                     &data,&bno
        dbne
                     Aspare, @doneg
                                              : if --spare!=-1
        bne.s
                     findneg
        move.1
                     &data,&spare
                                              : spare=daca
        subq. b
                     #7, &bno
                                              : bno-=6
                     &bno.&spare
#S007F.&spare
                                              ; spare>>=ono
        lsr.1
        andi.w
                                              : spare AND= mask
                     #8.&spare
        add.w
                                              ; spare+=9
        neg.w
                     &spare
      bra.s
                     ewrite
@fndneg subq.1
                     #7.&spare
                                              ; level-=8
Owrite Isl.w
                     &step, &spare
                                              ; level<<=step
                     &step
        SWAD
        add.w
                     Astep, Aspare
        SWAD
                     4step
                     &spare.&addr
        move.w
                                              : *addr=level
@quit
        endm
```

```
MACTO
         XVAL1
                      &addr.&step.&ptr.&data,&bno.&spare
         clr.w
                      &spare
         buf_rinc
                      áptr.ádata.ábno
                      Edata, Ebno
         buf_gec
         beq.s
                      @quit
                                               : if zero write
         moveq
                      #6.&spare
                                              ; set up count
         buf_get
                      &data, &bno
                                               : read sign
                      Odoneg
         bne.s
                                               ; if negative -> doneg
 @dopos
         buf_get
                      &data, &bno
         dbne
                      Espare, @dopos
                                              ; if --spare!=-1
         bne.s
                      @fndpos
         move. 1
                      &data,&spare
                                              ; spare=data
         subq.b
                      #7. Ebno
                                               ; bno-=6
         lsr.l
                      Ebno, Espare
                                              : spare>>=bno
                      #5007F. Espare
         andi.w
                                              ; spare AND= mask
         add.w
                      #8.&spare
                                              ; spare+=9
         bra.s
                     ewrite
 @fndpos neg.w
                     &spare
                                              ; bits-=bits
         addq.1
                     #7.&spare
                                              ; bits+=8
         bra.s
                     ewrite
 3doneg buf_get
                     &data, &bno
         dbne
                     &spare, @doneg
                                              : if --spare!=-1
         bne.s
                     @fndneg
         move.l
                     &data, &spare
                                              ; spare=data
         suba.b
                     #7,&bno
                                              : bno-=6
         lsr.l
                     Ebno, Espare
                                              : spare>>=bno
         andi.w
                     #$007F, &spare
                                              ; spare AND= mask
         add.w
                     #8. Aspare
                                              ; spare+#9
        neg.w
                     Sapare
        bra.s
                     Gwrite
@fndneg subq.l
                     #7, Espare
                                             : level-=8
@write lsl.w
                     &step,&spare
                                             ; level<<=step
Gquit
        move.w
                     Espare, &addr
                                             : *addr=level
        endm
        macro
        SKIPHUFF
                        &ptr.&data.&bno.&spare
                     &data, &bno
        buf_get
        beq.s
                     @quit
                                             ; if zero quit
        buf_get
                     &data, &bno
                                             ; skip sign
        moveq
                     #6,&spare
                                             ; set up count
9do
        buf_get
                    &data.&bno
        dbne
                    &spare, @do
                                             ; if --spare!=-1
        bne.s
                    9end
        subg.b
                    #7. &bno
                                             ; bno-=6
@end
        buf_rinc
                    aptr, adata, abno
                                             ; fill buffer
equit
        endm
```

```
macro
        XINTX
                     &bits, &addr. &step. &ptr. &data, &bno
    Note: half_q is missing
        buf_rinc
                     &ptr.&data.&bno
        move.l
                     &data.d0
                                              : result=data
        sub.b
                     &bits.&bno
                                              : dl-=bits-l
        subq.b
                     #1.&bno
                                              : d1-=1
        lsr.1
                     &bno.d0
                                              ; result>>=bno
        clr.1
                     d1
                                              ; d1=0
        bset
                     &bits,dl
                                              ; d1(bits)=1
        subq.l
                     #1.dl
                                              ; dl=mask
        btst
                     &bits.d0
                                              ; sigm?
; if positive goto pos
        beq.s
                     epos
        and.1
                     d1,d0
                                              ; apply mask leaving level
        neg.l
                     d0
                                              ; level-=level
        bra.s
                     econt
                                              ; goto cont
@pos
        and.l
                     d1,d0
                                              ; apply mask leaving level
@cont
        131.1
                     &step,d0
                                              ; level<<=step
                     d0.&addr
        move.w
                                              : 'addr = result
        endm
        macro
        XINT
                     &bits, &addr, &step, &ptr, &data, &bno
    Hardware compatable version: sign mag(lsb->msb)
        buf_rinc
                     &ptr.&data.&bno
        move.l
                     &data.d0
                                              ; result=data
        sub.b
                     &bits.&bno
                                              ; d1-=bits-1
        subq.b
                     #1.&bno
                                              : d1-=1
        lsr.1
                     &bno,d0
                                              : temo>>=bno
        clr.1
                     đl
                                              : result=0
        SWAD
                     £bno
                                              ; use free word
        move. v
                     &bits.&bno
                                              : bno=bnc.bits
        subq.w
                     #1,£bno
                                              ; count=bits-2
@shft
                     #1,d0
        lsr.1
                                              ; shift mab from Temp
        rox1.1
                     #1.dl
                                              : into 1sb of result
        dbf
                     &bno, @shft
                                             ; for entire magnitude
        swap
                     €puo
                                             : restore bno
        btst
                     #0.d0
                                              ; sign test
        beq.s
                     epos
                                              ; if positive -> pos
        neg.l
                     di
                                             : result = -result
@pos
        1s1.1
                     &step.dl
                                             ; result <<= step
                                             ; *addr=result
        move.w
                     dl.&addr
        endm
    Block data read/write:
    VOID, STILL. SEND, LPFSTILL
        macro
        VOID
                    &x_blk, &y_blk
        clr.w
                    (a2)
```

Engineering: KlicsCode: CompPict: KlicsDec2.a

```
addq.1
             &x_blk.a2
                                        : caddr+=x_blk
clr.w
             (a2)
adda.w
             &y_blk.a2
                                        ; caddr+=y_blk
clr.w
             (a2)
addq.1
             &x_blk.a2
                                        ; caddr+=x_blk
clr.w
             (a2)
endn
macro
             &x_blk. &y_blk. &step
STILL
XVAL0
             (a2), &step, a0, d6, d7, d0
addq.1
             &x_blk,a2
                                        ; caddr+=x_blk
XVALO
             (a2), &step, a0, d6, d7, d0
adda.w
                                        ; caddr+=y_blk
             &y_blk.a2
             (a2).&step.a0.d6.d7.d0
XVAL0
addq.1
             ax_blk.a2
                                        ; caddr+=x_blk
XVAL0
             (a2), &step, a0, d6, d7, d0
endm
macro
STILLSEND
             &x_blk. &y_blk. &step
XVAL1
             (a2), &step, a0, d6, d7, d0
addq.l
             &x_blk,a2
                                        ; caddr+=x_blk
             (a2), astep, a0, d6, d7, d0
XVAL1
adda.w
             ay_blk.a2
                                        ; caddr+=y_blk
XVAL1
             (a2), &step, a0, d6, d7, d0
addq.1
             &x_blk,a2
                                        : caddr+=x_blk
             (a2), &step, a0, d6, d7, d0
XVALL
endm
macro
SEND
             &x_blk,&y_blk,&step
XDELTA
             (a2),&step,a0,d6,d7.d0
                                        : caddr+=x_blk
addq.l
             ex_blk,a2
             (a2), &step, a0, d6, d7, d0
XDELTA
             &y_blk.a2
                                       : caddr+=y_blk
adda.w
             (a2),&step,a0,d6,d7,d0
XDELTA
                                        ; caddr+=x_blk
addq.1
             &x_blk.a2
            (a2), Estep, a0, d6, d7, d0
XDELTA
endm
macro
LPFSTILL
             &x_blk, &y_blk, &step, &bits -
             &bits, (a2), &step, a0, d6, d7
                                            ; ReadInt (at baddr)
XINT
                                              caddr+=x_blk
addq.1
             &x_blk,a2
                                              ReadInt
XINT
             &bits, (a2), &step, a0, d6, d7
                                             caddr+=y_blk
adda.w
             Ey_blk.a2
XINT
                                             ReadInt
             &bits, (a2), &step, a0, d6, d7
addq.l
                                            ; caddr+=x_blk
             Ex_blk.a2
                                            ; ReadInt
             &bits, (a2), &step, a0, d6, d7
```

```
Data skipping:
    SKIP4, STILLSKIP, SS_SKIP, SENDSKIP
..........
SKIP4
       FUNC
               EXPORT
        buf_rinc
                      a0.d6.d7
                                               ; fill buffer
        SKIPHUFF
                     a0.d6.d7.d0
        SKIPHUFF
                      a0.d6.d7,d0
        SKIPHUFF
                      a0.d6.d7,d0
        SKIPHUFF
                     a0.d6.d7.d0
        rts
        ENDFUNC
STILLSKIP FUNC
                      EXPORT
                                               ; BUF_INC
        buf_rinc
                      a0.d6.d7
        buf_get
                      d6.d7
                                               ; BUF_GET
        beq.s
                      eskl
                                               ; if 0 the STOP
        bsr
                     SKIP4
                                               ; BUF_INC
; BUF_GET
                     a0,d6,d7
        buf_rinc
eskl
        buf_get
                     d6.d7
                                               ; if 0 the STOP
        beq.s
                     esk2
        bsr
                      SKIP4
        buf_rinc
                      a0.d6.d7
                                               : BUF_INC
                                               : BUF_GET : if 0 the STOP
€sk2
         buf_get
                     d6,d7
        beq.s
                      @sk3
        bsr
                     SKIP4
                                               ; BUF_INC
        buf_rinc
                      a0.d6,d7
esk3
                                               ; BUP_GET
; if 0 the STOP
                     d6.d7
        buf_get
        beq.s
                     9nxt
                     SKIP4
        bsr
Onxt
        rts
        ENDFUNC
SS_SKIP FUNC.
                 EXPORT
                                               ; BUF_INC
; BUF_GET
        buf_rinc
                     a0,d6,d7
        buf_get
                     d6.d7
                                               ; if 0 then STOP
; BUF_GET
        beq.s
                     @sk1
        buf_get
                     d6.d7
                                               ; if \overline{1} then VOID
        bne.s
                     9skl
        bsr
                     SKIP4
                                              ; BUF_INC
; BUF_GET
                     a0,d6,d7
        buf_rinc
8sk1
        buf_get
                     d6.d7
                                               ; if 0 then STOP
        beq.s
                     esk2
                                               : BUF_GET
        buf_get
                     d6,d7
                                               ; if 1 then VOID
        bne.s
                     9sk2
        bsr
                     SKIP4
        buf_rinc
                     a0.d6.d7
                                               ; BUF_INC
@sk2
                     d6, d7
                                               ; BUF_GET
        buf_get
                                               ; if 0 then STOP
        beq.s
                     esk3
                                               ; BUF_GET
; if 1 then VOID
                     d6.d7
        buf_get
        bne.s
                     0sk3
        bar
                     SKIP4
                                               ; BUF_INC
; BUF_GET
        buf_rinc
                     a0,d6,d7
@sk3
        buf_get
                     d6,d7
                                               ; if \overline{0} then STOP
        beq.s
                     9DXE
        buf_get
                     d6.d7
                                               ; BUF_GET
```

- 663 -

Engineering:KlicsCode:CompPict:KlicsDec2.a

```
bne.s
                     enxt
                                              : if 1 then VOID
        bsr
                     SKIP4
3mxc
        rts
        ENDFUNC
SENDSKIP
            FUNC
                     EXPORT
        buf_rinc
                     a0.d6.d7
                                             : BUF_INC
        buf_get
                     d6.d7
                                             ; BUF_GET
        beq.s
                     @skl
                                             ; if 0 the STOP
        buf_gec
                     d6.d7
                                             ; BUF_GET
        beq.s
                     9sk0
                                             ; if 0 then STILLSEND
        buf_get
                     d6, d7
                                             ; BUF_GET
        beq.s
                     esk1
                                             ; if 0 then VOID
@sk0
                     SKIP4
        bsr
        buf_rinc
                     a0,d6,d7
                                             ; BUF_INC
0skl
        buf_get
                     d6.d7
                                             ; BUP_GET
                                             ; if 0 the STOP
        beq.s
                     8sk3
        buf_get
                     d6, d7
                                             ; BUP_GET
        beq.s
                     0sk2
                                             : if 0 then STILLSEND
        buf_get
                     d6, d7
                                             ; BUP_GET
        beq.s
                     esk3
                                             : if 0 then VOID
esk2
        bsr
                     SKIP4
        buf_rinc
                    a0.d6.d7
                                             ; BUF_INC
8sk3
        buf_get
                     d6.d7
                                             ; BUF_GET
        beq.s
                     Gsk5
                                             ; if 0 the STOP
        buf_get
                     d6.d7
                                             ; BUF_GET
        beq.s
                     9sk4
                                             : if 0 then STILLSEND
                     d6.d7
        buf_get
                                             ; BUF_GET
        beq.s
                     0sk5
                                             ; if 0 then VOID
esk4
        bsr
                    SKIP4
        buf_rinc
                    a0.d6,d7
                                             ; BUF_INC
0sk5
        buf_get
                    d6,d7
                                             : BUF_GET
        beq.s
                    @nxc
                                               if 0 then STOP
                    d6.d7
        buf_get
                                             ; BUF_GET
        beq.s
                    esk6
                                             ; if 0 then STILLSEND
        buf_get
                    d6.d7
                                             : BUF_GET
        beq.s
                    Onxt.
                                             ; if 0 then VOID
esk6
        bsr
                    SKIP4
enxt
        rts
        ENDFUNC
********************
    Octave Processing:
    DOSTILLO, DOSENDO, DOSTILLI,
    DOVOIDI, DOSTILLSENDI, DOSENDI
DOSTILLO
```

FUNC EXPORT

;

```
Engineering:KlicsCode:CompPict:KlicsDec2.a
```

```
buf_rinc
buf_get
                        a0.d6.d7
                                                  : BUF_INC
                       d6.d7
                                                  : BUF_GET
          bne.s
                        @still
                                                  ; if I the STILL
          rts
 ?still move.1
                       a1,a2
                                                 : caddr=baddr
          STILL
                       44.d5.d3
          XVAL0
                       (a2),d3,a0.d6,d7,d0
          addq.1
                       #4.a2
                                             : caddr+=x_blk
          XVALO
                       (a2),d3,a0,d6,d7,d0
          adda.w
                       d5.a2
                                             : caddr+=y_blk
          XVALO
                       (a2),d3,a0,d6,d7.d0
          addq.1
                       #4.a2
                                             : caddr+=x_blk
         XVALO
                       (a2),d3,a0,d6,d7,d0
          bsr
                       STILLSKIP
         rts
         ENDFUNC
 DOSENDO FUNC
                  EXPORT
         buf_rinc
buf_get
                       a0.d6.d7
                                                 : BUF_INC
                       d6.d7
                                                 ; BUF_GET
         bne.s
                       econt
                                                 ; if I then continue
         rts
3cont
         move.1
                      al, a2
                                                : caddr=baddr
         buf_get
                                                ; BUF_GET ; if 0 then STILLSEND
                      d6,d7
         beq.w
                      955
         buf_get
                      d6.d7
                                                ; BUP_GET
         beq.w
                      Ovd
                                                ; if 0 then VOID
         SEND
                      #4.d5.d3
         XDELTA
                      (a2).d3.a0.d6.d7,d0
                      #4,a2
(a2).d3.a0,d6,d7,d0
         addq.1
                                                ; caddr+=x_blk
         XDELTA
                      d5,a2
(a2),d3,a0,d6,d7,d0
         adda.w
                                                ; caddr+=y_blk
         XDELTA
         addq.1
                      #4,a2
                                                : caddr+=x_blk
         XDELTA
                      (a2).d3,a0,d6,d7,d0
        bsr
                      SENDSKIP
        rts
655
        :STILLSEND #4,d5,d3
        XVAL1
                      (a2),d3,a0,d6,d7,d0
        addq.1
                      #4,a2
                                           : caddr+=x_blk
        XVAL1
                      (a2),d3,a0,d6,d7,d0
        adda.w
                     d5, a2
                                           : caddr+=y_blk
        XVAL1
                      (a2), d3, a0, d6, d7, d0
        addq.1
                     #4.42
                                           ; caddr+=x_blk
        XVAL1
                     (a2),d3,a0,d6,d7.d0
        bsr
                     SS_SKIP
        rts
bve
        : VOID
                     #4.d5
```

ı

```
Engineering:KlicsCode:CompPict:KlicsDec2.a
          clr.w 🕳
                        (a2)
           addq.1
                        #4.a2
                                             ; caddr+=x_5lk
          clr.w
                        (a2)
          adda.w
                        d5,a2
                                            ; caddr+=y_blk
          clr.w
                        (a2)
          addq.1
                        44.a2
                                             ; caddr+=x_blk
          clr.w
                        (a2)
          rts
          ENDFUNC
          macro
          DOSTILL1
                       &addr
          buf_get
                       d6.d7
                                                 ; BUF_GET
          beq.w
                       enext
                                                 ; if 0 the STOP
          move.1
                       al,a2
                                                 : caddr=baddr
          add.1
                       &addr,a2
                                                 ; caddr+=addrs(1)
          STILL
                       #4.d5,d4
          bsr
                       STILLSKIP
          buf_rinc
                       a0,d6,d7
                                                : BUF_INC
 @next
          endm
          macro
          DOVOID1
                      &addr
          move.1
                       a1.a2
                                                : caddr=baddr
          add.1
                      &addr,a2
                                                ; caddr+=addrs(1)
         VOID
                      #4.d5
         endm
         macro
         DOSTILLSEND1
                          &addr
         buf_get
                      d6.d7
                                               ; BUF_GET
; if 0 the STOP
         beq.w
                      @next
         move.1
                      al.a2
                                                : caddr=baddr
         add. 1
                      &addr,a2
                                                ;. caddr+=addrs(1)
         buf_get
                      d6,d7
                                               ; BUF_GET ; if 0 then STILLSEND
         beg.s
                      935
         VOID
                      #4.d5
         bra
                      Onext
935
         STILLSEND
                      #4, d5, d4
        bsr
                     SS_SKIP
        buf_rinc
                     a0,d6,d7
                                               : BUF_INC
enext
        endm
DOSTILL2
            FUNC
                     EXPORT
        buf_rinc
                     a0,d6,d7
                                              : BUF_INC
        buf_get
                     d6,d7
                                              ; BUF_GET
; if 1 the CONT
        bne.s
                     Scont
        rts
@cont
        move.1
                     a1.a2
                                              : caddr=baddr
```

buf_rinc DOSEND1

DOSEND1

a0,d6,d7

4 (a3)

8(a3)

```
Engineering:KlicsCode:CompPict:KlicsDec2.a
          add.l
                        (a3).a2
                                                   : caddr+=addrs(0)
          STILL
                        48.d5.d3
          swap
          exg
                        d4.a5
          buf_rinc
                        a0.d6.d7
                                                   : BUF_INC
          DOSTILLI
                        4 (a3)
          DOSTILLI
                        8(a3)
          DOSTILLI
                        12(a3)
          DOSTILL1
                        16(a3)
          SWAD
                        ₫5
          exg
                       d4, a5
          rts
          DOSEND1
                       &addr
          buf_ge:
                       d6,d7
                                                  ; BUF_GET
          beq.w
                       enext
                                                  ; if 0 the STOP
         move.1
                       al.a2
                                                   : caddr=baddr
          add.l
                       &addr,a2
                                                    caddr+=addrs[1]
         buf_get
                       d6.d7
                                                  ; BUP_GET
          beq.w
                       988
                                                  ; if 0 then STILLSEND ; BUP_GET
         buf_get
                       d6, d7
         beq.w
                       evd
                                                  ; if 0 then VOID
         SEND .
                       #4.d5.d4
         par
                       SENDSKIP
         bra
                       Orinc
evd
         VOID
                       #4.d5
         bra
                       enext
999
         STILLSEND
                       #4.45.44
                      SS_SKIP
a0.d6.d7
         bsr
grinc
         buf_rinc
                                                 ; BUF_INC
ênext
         endim
DOSEND2 FUNC
                  EXPORT
        buf_rinc
buf_get
                      a0,d6,d7
                                                 : BUF_INC
                      d6.d7
                                                 : BUF_GET
         bne.s
                      9cont
                                                 : if I the CONT
@nxt
         rts
Gcont
        move.1
                      al, a2
                                                 ; caddr=baddr
                      (a3),a2
d6,d7
        add.l
                                                 ; caddr+=addrs(0)
        buf_get
                                                 : BUF_GET
        beg.w
                      955
                                                ; if 0 then STILLSEND
; BUF_GET
; if 0 then VOID
        buf_get
                      d6,d7
        beq.w
                      0 vd
*** SEND ***
        SEND
                      48,d1,d3
```

; BUF_INC

```
Engineering:KlicsCode:CompPict:KlicsDec2.a
                   = 12(a3)
16(a3)
        DOSEND1
        DOSEND1
        rts
*** STILLSEND ***
        STILLSEND #8.d1.d3
                    a0.d6.d7
        buf_rinc
                                                : .BUF_INC
        DOSTILLS END1
                          4(a3)
        DOSTILLSEND1
DOSTILLSEND1
                          8(a3)
                          12(a3)
        DOSTILLSEND1
                          16(a3)
        rts
*** VOID ***
3vd
        voin
                      #8,d1
        DOVOID1
                      4(a3)
        DOVOID1
                     8(a3)
        DOVOID1
                      12(a3)
        DOVOID1
                     16(a3)
        rts
        ENDFUNC
        macro
        UVSTILL0
    Low_Pass
        move.l
                     a1,a2
                                               ; caddr=baddr
                     #4.d5,d2,d4
        LPFSTILL
    Sub-band gh
        addq.1
                     #2,a1
                                               ; baddr+=2 (gh band)
        bsr
                     DOSTILLO
    Sub-band hg
        subq.1
                     #2,a1
                                               ; baddr-=2 (hh band)
        add.1
                     a4.al
DOSTILLO
                                               ; caddr+=1 row (hg band)
        bsr
    Sub-band gg
        addq.l
                     #2.al
                                               ; baddr+=2 (gg band)
        bsr
                     DOSTILLO
        sub. 1
                     a4,a1
                                               ; caddr-=1 row (gh band)
        addq.1
                     #6,a1
                                               ; (2+) addr(0)+=x_inc
        endm
        macro
        UVSENDO
   Low_Pass
                                              : BUF_INC
: BUF_GET
: if 0 then process subbands
        buf_rinc
                     a0.d6.d7
        buf_get
                     d6,d7
        beq.w
                     @ subs
```

```
Engineering: KlicsCode:CompPict:KlicsDec2.a
         move.l 🕳
                       al.a2
                                                : caddr=baddr
         SEND
                       #4.d5.d2
     Sub-band gh
 3subs
         addq.1
                       #2.al
                                                : baddr-=2 (gh band)
                      DOSENDO
     Sub-band ng
         subq.l
                      #2.al
                                                ; baddr-=2 (hh band)
         add.l
                      a4.a1
                                                ; caddr+=1 row (hg band)
         bsr
                      DOSENDO
     Sub-band gg
         addq.1
                      #2.al
                                                ; baddr+=2 (gg band)
         bar
                      DOSENDO
         sub. 1
                      a4.al
                                                : caddr-=1 row (gh band)
         aridq.l
                      #6,a1
                                                : (2+) addr(0)+=x_inc
    Decoder functions:
    Klics2D1Still, Klics2D1Send
Klics2D1Still FUNC
                          EXPORT
    Klics2D1Still(short *dst, long size_x, long size_y, long lpfbits, short *norms
PS
        RECORD
dst
        DS.L
size_x DS.L
                      1
size_y
        DS.L
lpfbits DS.L
norms
        DS.L
        DS.L
ptr
dara
        DS.L
onc
        DS.L
        ENDR
LS
        RECORD
                     0.DECR
x_lim
        DS.L
                                               : x counter termination
                                                                             row_start+
x_linc
        DS.L
                                               ; x termination increment
                                                                             l row
y_inc0
        DS.L
                                               ; y counter increment
                                                                              4 rows
y_incl
        DS.L
                                               ; y counter increment ; y counter termination
                                                                             7 rows
y_lim
LSize
        DS.L
                                                                             area
        EQU
        ENDR
   d0/d1 - spare
   d2 - step 0 (HH)
d3 - step 0
d4 - lpfbits
   d5 - y_blk
d6 - data
                 (bit stream)
   d7 - bno
                 (bit pointer)
```

```
Engineering: KlicsCode:CompPict:KlicsDec2.a
```

```
a0 - ptr
                   (bit buffer)
     al - baddr (block address)
     a2 - caddr
                  (coeff address)
     al - x_lim
     a4 - x_linc
     a5 - y_inc0
         1 ink
                       a6, #LS.LSize
                                                   : locals
         movem.1
                       d4-d7/a3-a5, -(a7)
                                                   ; store registers
     Load Bit Buffer
         move.1
                        PS.data(a6),a0
                                                   ; a0=&data
         move.1
                        (a0),d6
                                                   ; data= a0
         move.1
                       PS.bno(a6),a0
                                                   ; a0=&mask
         move.1
                        (a0).d7
                                                  ; mask=*a0
         move.l
                        PS.ptr(a6),a0
                                                  ; a0=&ptr
         move.1
                        (a0).a0
                                                  ; a0=ptr
     Set Up Block Counters
         move.1
                       PS.dst(a6),al
                                                  ; al=image
         move.1
                       PS.size_x(a6),d0
                                                  ; d0=size_x
         add.l
                       d0,d0
                                                  ; in shorts
         move.l
                       d0.LS.x_linc(a6)
                                                  ; x_linc=1 row
         move.1
                       PS. size_y(a6),d1
                                                  ; dlasize_y
         muls.w
                       d0, d1
                                                    dl==d0 (area)
         add.l
                       al,d1
                                                   : dl+=image
                       dl, LS.y_lim(a6)
         move.1
                                                  ; y_lim=d1 ; d2=d0 (1 row)
                       d0, d2
         move.1
         add.l
                       d0, d0
                                                    d0*=2 (2 rows)
         move.1
                       d0.d5
                                                  ; y_blk=d0
         subg. 1
                       $4,d5
                                                  ; y_blk-=x_blk
         add.l
                       d0.d0
                                                  ; d0*=2 (4 rows)
                                                  ; y_inc0=d0
; d0*=2 (8 rows)
; d0-=d2 (7 rows)
         move.1
                       d0.LS.y_inc0(a6)
         add.1
                       d0,d0
         sub.1
                       d2,d0
         move.1
                       d0.LS.;_incl(a6)
                                                  ; y_incl=d0
         move.1
                       PS.norms(a6),a2
                                                  : GetNorm pointer
         move.l
                       (a2),d2
                                                  : read normal
                      4(a2),d3
PS.lpfbits(a6),d4
         move.1
                                                  ; read normal
         move.1
                                                  : read lpfbits
         move.l
                       LS.x_linc(a6).a4
                                                  : read x_linc
         move.1
                      LS.y_inc0(a6), a5
                                                  ; read y_inc0
0y
         move.l
                      a4,a3
                                                  ; x_lim=x_linc
         add.l
                      a1.a3
                                                  : x_lim+=baddr
0 x
         UVSTILL0
                                                 : process UV block 0,0 : process UV block 1,0
         UVSTILL0
                                                 : (2) addr[0]+=y_inc
: (2+) addr[0]-limit?
         add.l
                      a5,a1
         cmp.1
                      LS.y_lim(a6),al
         bge.w
                      Plast
                                                  ; if half height
         sub. 1
                      #16,a1
                                                  ; pointer=blk(0,1)
         UVSTILLO
                                                 ; process UV block 0,1 ; process UV block 1,1
         UVSTILL0
Olast
         sub.1
                      a5,al
                                                 ; (2) addr(0)+=y_inc
         cmp.1
                      a3.a1
                                                 ; (2+) addr(0)-limit?
         blt.w
                      Øх
                                                 ; (4) if less then loopX
         add.1
                      LS.y_incl(a6),al
                                                 ; (2+) addr(0)+=y_inc
                                                 ; (2+) addr[0]-limit?
; (4) if less then loopy
         CMD.l
                      LS.y_lim(a6),a1
        blt.w
```

```
Save Bit Buffer
                     PS.data(a6),a2
        move.1
                                              ; spare=&data
                                              : update data
        move.1
                     d6. (a2)
        move.1
                     PS.bno(a6), a2
                                              ; spare=&bno
        move. 1
                     d7, (a2)
                                              : update bno
                                              ; spare=&ptr
        move 1
                     PS.ptr(a6),a2
        move.l
                     a0.(a2)
                                              : update ptr
        movem.1
                     (a7)+,d4-d7/a3-a5
                                              : restore registers
        unlk
                                              ; remove locals
                     a6
        rts
                                              : return
        ENDFUNC
Klics2D1Send FUNC EXPORT
    Klics2D1Send(short *dst. long size_x, long size_y, short *norms, unsigned long
PS
        RECORD
dst
        DS.L
size_x DS.L
size_y
        DS.L
nome
        DS.L
per
        DS.L
data
        DS.L
pno
        DS.L
        ENDR
LS
        RECORD
                     0, DECR
x_lim
        DS.L
                                              ; x counter termination
                                                                           row_start+
x_linc DS.L
                                              ; x termination increment
                                                                           1 row
                     1
y_inc0
        DS.L
                                              ; y counter increment
                                                                           4 rows
y_incl
        DS.L
                     1
                                              ; y counter increment
                                                                           7 rows
y_lim
        DS.L
                                              ; y counter termination
                                                                           area
LSize
        EQU
        ENDR
    d0/d1 - spare
    d2 - step 0 (HH)
d3 - step 0
    d4 - y_inc0
d5 - y_blk
    d6 - data
                 (bit stream)
    d7 - bno
                 (bit pointer)
    a0 - pt:
                 (bit buffer)
    al - baddr (block address)
    a2 - caddr
                (coeff address)
    a3 - x_lim
    a4 - x_linc
a5 - y_lim
                    a6, #LS. LSize
                                             ; locals
        link
                    d4-d7/a3-a5,-(a7)
                                             ; store registers
        movem.1
    Load Bit Buffer
        move. 1
                     PS.daca(a6),a0
                                             ; a0=&data
        move.1
                     (a0), G6
                                             ; data="a0
        move.1
                     PS.bno(a6), a0
                                             ; a0=&mask
        move.1
                     (a0).d7
                                             ; mask=*a0
```

```
move. 1
                       PS.ptr(a6), a0
                                                   : a0=&ptr
         move. 1
                       (a0).a0
                                                   : a0=pcr
     Set Up Block Counters
                       PS.dst(a6),al
         move. 1
                                                  : al=image
         move. 1
                       PS.size_x(a6).d0
                                                  : d0=size_x
         add.l
                       d0,d0
                                                   : in shorts
         move.1
                       d0.LS.x_linc(a6)
                                                   ; x_linc=1 row
         move.1
                       PS. size_y(a6),d1
                                                  : dl=size_y
         muls.w
                                                   : d1*=d0 (area)
                       d0.d1
         add.l
                       al,dl
                                                   ; dl+=image
         move.1
                       dl.LS.y_lim(a6)
                                                   ; y_lim=dl
         move.l
                       d0.d2
                                                  ; d2=d0 (1 row)
         add.1
                       d0,d0
                                                  : d0 =2 (2 rows)
         move.1
                       d0.d5
                                                  ; copy to d5
         subq.l
                       #4.d5
                                                  ; subtract x_blk
         add.l
                       40,40
                                                  ; d0*=2 (4 rows)
         move. 1
                       d0, LS.y_inc0(a6)
                                                  ; y_inc0=d0
         add.1
                                                  ; d0*=2 (8 rows)
; d0-=d2 (7 rows)
                       d0.d0
         sub. 1
                       d2,d0
                       d0.LS.y_incl(a6)
         move.1
                                                  ; y_incl=d0
         move.1
                       PS.norms(a6),a2
                                                  ; GetNorm pointer
         move.1
                       (a2),d2
                                                  ; read normal
         move.l
                       4(a2).d3
                                                . : read normal
         move.1
                                                 ; read x_linc
; read y_inc0
                       LS.x_linc(a6),a4
         move.1
                      LS.y_inc0(a6),d4
LS.y_lim(a6),a5
                                                 ; read y_lim
         move.1
Øy
         move.1
                       a4, a3
                                                  ; x_lim=x_linc
; x_lim+=baddr
         add.l
9×
         UVSENDO
                                                  ; process UV block 0.0 ; process UV block 1.0
         UVSENDO
         add.l
                      d4,al
                                                  ; (2) addr(0]+=y_inc; (2) addr(0]-limit?
         cmp.1
                      a5.a1
         bge.w
                      Glast
                                                  ; if half height
         sub. 1
                      #16,a1
                                                 ; pointer=blk(0,1)
        UVSENDO
                                                 ; process UV block.D.1
        UVSENDO
                                                 : process UV block 1,1
Glast
        sub.1
                      d4, a1
                                                 : (2) addr(0)+=y_inc
        cmp.l
                      a3,a1
                                                 ; (2) addr[0]-limit?
        blt.w
                                                 ; (4) if less then loopX
        add.1
                      LS.y_incl(a6),al
                                                 ; (2+) addr[0]+=y_inc
        cmp.l
                      a5,al
                                                 ; (2) addr[0]-limit?
; (4) if less then loopy
        blt.w
                      @y
    Save Bit Buffer
                      PS.data(a6),a2
        move. 1
                                                 ; spare=&data
        move.1
                      d6, (a2)
                                                 ; update data
        move.1
                      PS.bno(a6),a2
                                                 ; spare=ibno
        move.1
                      d7, (a2)
                                                 ; update bno
                      PS.ptr(a6),a2
a0,(a2)
        move.i
                                                 ; spare=&ptr
        move.1
                                                 : update ptr
        movem.1
                      (a7)+,d4-d7/a3-a5
                                                 ; restore registers
        unlk
                      a6
                                                 ; remove locals
        rts
                                                 ; return
        ENDPUNC
```

- 672 -

```
Klics3D2Still FUNC
                         EXPORT
    Klics3D2Still(short *dst. long size_x. long size_y. long lpfbits, short *norms
PS
        RECORD
        DS.L
dst
                     1
size_x DS.L
        DS.L
size_y
lpfbits DS.L
        DS.L
norms
DEE
        DS.L
data
        DS.L
Fuc
        DS.L
sub_tab DS.L
        ENDR
LS .
        RECORD
                     0. DECR
y_blk0 DS.L
                                               ; y inter-block increment
                                                                             2 rows - 4
y_blk1 DS.L
                                                                             4 rows - 8
                     1
                                               ; y inter-block increment
x_inc
        DS.L
                                               ; x counter increment
                                                                             16
x_lım
        DS.L
                     1
                                               ; x counter termination
                                                                             row_start+
x_linc
        DS.L
                                               : x termination increment
                                                                             1 row
y_inc
        DS.L
                                               ; y counter increment
                                                                             7 rows
y_lim
LSize
        DS.L
                                               ; y counter termination
                                                                             area
        EQU
        ENDR
    d0/d1 - spare
    d2 - step 2HH
d3 - step 1
    d4 - step 0/lpfbits
    d5 - y_blk0,y_blk1
d6 - data (bit stream)
    d7 - bno
                 (bit pointer)
    a0 - ptr
                 (bit buffer)
    al - baddr (block address)
a2 - caddr (coeff address)
    a3 - addrs (tree addresses)
    a4 - x_lim (x counter termination)
    a5 - lpfbits/step 0
                     a6, #LS.LSize
                                              ; locals
                     d4-d7/a3-a5, - (a7)
        movem.l
                                              ; store registers
    Load Bit Buffer
        move.1
                     PS.data(a6).aC
                                              : a0=&data
        move.1
                     (a0),d6
                                              : data="a0
        move.l
                     PS.bno(a6), a0
                                              ; a0=&mask
        move.1
                     (a0),d7
                                              ; mask=*a0
        move.1
                     PS.ptr(a6), a0
                                              ; a0=&ptr
        move.1
                     (a0),a0
                                              ; a0=ptr
    Set Up Block Counters
        move.1
                                              : al=image
                     PS.dst(a6),al
        move.l
                     PS.size_x(a6),d0
                                              ; d0=size_x
        move.1
                     #16, LS. x_inc(a6)
                                              ; save x_inc
        add.1
                     d0.d0
                                              ; in shorts
        move.1
                     d0.LS.x_linc(a6)
                                              ; x_linc=l row
        move.l
                     PS.size_y(a6),dl
                                              ; dl=size_y
        muls.w
                     d0.d1
                                              ; d1 = d0 (area)
```

```
Engineering:KlicsCode:CompPict:KlicsDec2.a
```

```
add.1 =
                        al,d1
                                                   : dl+=image
          move.l
                        dl.LS.y_lim(a6)
                                                   ; y_lim=d1
          move.l
                        40,42
                                                   : d2=d0 (1 row)
          add.l
                        d0,d0
                                                   : d0*=2 (2 rows)
          move.1
                        d0,d5
                                                   : copy to d5
          subq. 1
                        #4.d5
                                                   ; y_blk: subtract x_blk
          move.1
                        d5.LS.y_blk0(a6)
                                                   : save y_blk0
          add. 1
                        d0,d2
                                                    d2+=d0 (3 rows)
          add.l
                       d0.d0
                                                   ; d0°=2 (4 rows)
          move.l
                        d0.d4
                                                   ; copy to d5
          subq.l
                        #8,d4
                                                   : y_blk: subtract x_blk
                       d4, LS.y_blk1(a6)
          move.1
                                                  ; save y_blk1
; d0+=d2 (7 rows)
          add.l
                       d2, d0
          move.1
                       d0, LS.y_inc(a6)
                                                  : y_inc=d0
          move.1
                       PS.norms(a6),a2
                                                  ; GetNorm pointer
          move.l
                       (a2).d2
                                                  : read normal
          move.l
                        4(a2),d3
                                                  : read normal 1
          move.1
                       8(a2),a5
                                                  ; read normal 0
         move.1
                       PS. lpfbits(a6), d4
                                                  ; read lpfbits
          swap
                       d5
                                                  ; y_blk=00XX
         move.1
                       LS.y_blk1(a6),d0
                                                  ; read y_blk1
         move.w
                       d0.d5
                                                    d5=y_b1k0/1
         move.1
                       PS.sub_tab(a6),a3
                                                  : a3=addrs
Øу
         move.l
                       LS.x_linc(a6),a4
                                                  : x_lim=x_linc
         add. 1
                       a1, a4
                                                  ; x_lim+=baddr
     Low_Pass
θx
         move.1
                       al, a2
                                                 ; caddr=baddr
         LPPSTILL
                       #8.d5.d2,d4
     Sub-band gh
         bar
                       DOSTILL2
         add.1
                       #20,a3
    Sub-band hg
         bsr
                      DOSTILL2
         add.1
                      #20,a3
    Sub-band gg
         bsr
                      DOSTILL2
         sub. 1
                      #40.a3
         add.1
                      #16, al
                                                 ;"(2) addr(0)+=x_inc
        cmp.1
                                                (2) addr[0]-limit?

(4) if less then loopX

(2+) addr[0]-sy_inc

(2+) addr[0]-limit?
                      a4.al
        blt.w
                      θx
        add.1
                      LS.y_inc(a6),a1
        cmp.1
                      LS.y_lim(a6),a1
        blt.w
                                                ; (4) if less then loopy
    Save Bit Buffer
Gend
        move.1
                      PS.data(a6),a2
                                                ; spare=&data
        move.l
                      d6, (a2)
                                                  update data
                     PS.bno(a6), a2
        move.1
                                                : spare=&bno
        move.1
                     d7, (a2)
                                                ; update bno
        move.1
                     PS.ptr(a6), a2
                                                ; spare=&ptr
        move.1
                     a0,(a2)
                                                ; update ptr
```

move. 1

Engineering:KlicsCode:CompPict:KlicsDec2.a

```
movem.1
                      (a7)+,d4-d7/a3-a5
                                                 : restore registers
         unlk
                       a6
                                                 : remove locals
         rts
                                                 : return
         ENDFUNC
Klics302Send
                 FUNC
                           EXPORT
    Klics3D2Send(short *dst. long size_x, long size_y, short *norms, unsigned long
PS
         RECORD
                       8
dst
         DS.L
                       1
size_x
         DS.L
                      1
Size_y
         DS.L
norms
         DS.L
ptr
         DS.L
data
         DS.L
puo
         DS.L
sub_tab DS.L
         ENDR
LS
         RECORD.
                      0, DECR
y_blk0 DS.L
                                                 ; y inter-block increment
                                                                                2 rows - 4
y_blk1
         DS.L
                                                 ; y inter-block increment
                                                                                4 rows - 8
x_inc
         DS.L
                                                 ; x counter increment
                                                                                16
                                                 ; x counter termination
x_lim
         DS.L
                                                                                row_start+
x_linc DS.L
                                                 ; x termination increment
                                                                                1 row
y_inc
         DS.L
                                                                                7 rows
                                                 ; y counter increment
y_lim
                                                ; y counter termination
         DS.L
                                                                                area
LSize
         EQU
         ENDR
    d0 - spare
d1 - y_blk1
d2 - step 2HH
    d3 - step 1
    d4 - step 0
d5 - y_blk0
d6 - data
d7 - bno
                (bit stream)
                  (bit pointer)
    a0 - ptr (bit burrer
a1 - baddr (block address)
-2 - caddr (coeff address)
    a3 - addrs
                 (tree addresses)
    a4 - x_lim (x counter termination)
                      a6, #LS.LSize
d4-d7/a3-a5, -(a7)
        link
                                                ; locals
        movem.1
                                                : store registers
    Load Bit Buffer
        move.1
                      PS.data(a6),a0
                                                ; a0=&data
        move.l
                      (a0),d6
                                                ; data=*a0
        move.1
                      PS.bno(a6), a0
                                                : a0=&mask
        move.l
                      (a0),d7
                                                : mask=*a0
        move.1
                      PS.ptr(a6),a0
                                                : a0=Eptr
        move.l
                      (a0),a0
                                                ; a0=ptr
    Set Up Block Counters
                      PS.dsc(a6).al
```

; al=image

ł

```
move.1
                       PS.size_x(a6).d0
                                                  ; d0=size_x
         move.1
                       *16.LS.x_inc(a6)
                                                  : save x_inc
         add.1
                       d0,d0
                                                   ; in shorts
         move. 1
                       d0, LS.x_linc(a6)
                                                    x_linc=l row
         move. 1
                       PS.size_y(a6).dl
                                                    dl=size_y
         muls.w
                       d0.d1
                                                    d1 ad0 (area)
         add. 1
                       al.dl
                                                  : dl+=image
         move.l
                       dl.LS.y_lim(a6)
                                                  : y_lim=dl
         move.1
                                                  ; d2=d0 (1 row)
                       d0.d2
         add.l
                       d0,d0
                                                  : d0*=2 (2 rows)
                                                    copy to d5
y_blk: subtract x_blk
         move.l
                       d0.d5
         subg. 1
                       #4.d5
         move. 1
                       d5.LS.y_b1k0(a6)
                                                    save y_blk0
         add.1
                                                    d2+=d0 (3 rows)
d0*=2 (4 rows)
                       d0.d2
         add.1
                       d0, d0
         move. 1
                                                    copy to d5
y_blk: subtract x_blk
                       d0,d4
         subq.1
                       #8.d4
         move. 1
                       d4, LS.y_blk1 (a6)
                                                    save y_blk1
d0+=d2 (7 rows)
         add. 1
                       d2,d0
                                                  ; y_inc=d0
         move.l
                       d0.LS.y_inc(a6)
         move.1
                       PS.norms(a6),a2
                                                  ; GetNorm pointer
         move.1
                       (a2).d2
                                                  ; read normal
         move.1.
                       4(a2),d3
                                                  ; read normal 1
         move.1
                       8(a2),d4
                                                  ; read normal 0
                       LS.y_blk1(a6),d1
         move.1
                                                  ; read y_blk1
         move.1
                       PS. sub_tab(a6), a3
                                                  ; a3=addrs
         move.l
₽y
                      LS.x_linc(a6),a4
                                                  ; x_lim=x_linc
         add.1
                       al, a4
                                                  ; x_lim+=baddr
    Low_Pass
                                                 ; BUF_INC
; BUF_GET
; if 0 then process subbands
                      a0.d6.d7
в×
         buf_rinc
         buf_get
                      d6, d7
         beq.w
                      0 subs
         move. 1
                                                  ; caddr=baddr
                      al,a2
         SEND
                      #8,d1,d2
    Sub-band gh
@ subs
         bar
                      DOSEND2
         add.1
                       #20.a3
    Sub-band hg
         bsr
                      DOSEND2
         add. 1
                      #20,a3
    Sub-band gg
                      DOSEND2
        bsr
        sub.1
                      #40,a3
        add.1
                      #16,a1
                                                 ; (2) addr[0]+=x_inc
                                                 ; (2) addr[0]-limit?
; (4) if less then loopX
         cmp.1
                      a4,al
        blt.w
                      0x
         add.l
                      LS.y_inc(a6),al
                                                 ; (2+) addr(0)+=y_inc
         Cmp.1
                                                 ; (2+) addr(0)-limit?
                      LS.y_lim(a6),al
        blt.w
                                                 ; (4) if less then loopY
    Save Bit Buffer
```

- 676 -

₹end	move.1 move.1 move.1 move.1 move.1	PS.data(a6),a2 d6,(a2) PS.bno(a6),a2 d7,(a2) PS.ptr(a6),a2 a0,(a2)	: spare=&data : update data : spare=&bno : update bno : spare=&ptr : update ptr
•	movem.l unik rts ENDFUNC	(a7)+,d4-d7/a3-a5 a6	<pre>: restore registers : remove locals : return</pre>
	END		

```
/*****
     C Copyright 1993 KLICS Limited
     All rights reserved.
     Written by: Adrian Lewis
  Importing raw Klics binary files
     Stand-alone version
#include
               'Bits3.h'
#include
               'Klics.h'
#include
               *RlicsHeader.h*
typedef char
                    Boolean:
/* If bool true the negate value */
#define negif(bool, value)
                                 ((bool)?-(value):(value))
extern void
                    HaarBackward();
                   Daub4Backward(short *data.int size(2).int oct_src);
TestTopBackward(short *data.int size(2).int oct_src);
extern void
extern
         void
                   TestBackward(short *data.int size(2),int oct_src); KLICSDCHANNEL(short *dst. long octs. long size_x, long size_y, long
extern void
extern void
/* Use the bit level file macros (Bits2.h) */
/* buf_use: */
/* Huffman decode a block */
#define HuffDecLev(lev,buf) \
     lev(0)=HuffDecode(buf); \
     lev(1)=HuffDecode(buf); \
     lev(2)=HuffDecode(buf); \
     lev(3)=HuffDecode(buf);
/* Fixed length decode block of integers */
#define IntDecLev(lev,lpf_bits,buf) \
     lev(0)=IntDecode(lpf_bits.buf); \
     lev[1] = Int Decode(lpf_bits, buf); \
     lev(2)=IntDecode(lpf_bits.buf); \
     lev(3)=IntDecode(lpf_bits,buf);
/* Reverse quantize difference block */
*define RevOntDelta(new,old,lev,shift) \
    new[0]=old[0]+(lev[0]<<shift)+(lev[0]!=0?negif(lev[0]<0,(l<<shift)-l>>1):0); \
    new[1] =old[1]+(lev[1]<<shift)+(lev[1]!=0?negif(lev[1]<0,(1<<shift)-1>>1):0); \
new[2] =old[2]+(lev[2]<<shift)+(lev[2]!=0?negif(lev[2]<0,(1<<shift)-1>>1):0); \
    new(3)=old(3)+(lev(3)<<shift)+(lev(3)!=0?negif(lev(3)<0,(1<<shift)-1>>1):0);
/* Reverse quantize block */
#define RevQnt(new,lev,shift) \
    new(0] = (lev(0) << shift) + (lev(0)! = 0?negif(lev(0) < 0, (1 << shift) -1>>1):0); \
    new[1]=(lev[1]<<shift)+(lev[1]!=0?negif(lev[1]<0,(1<<shift)-1>>1):0); \
new[2]=(lev[2]<<shift)+(lev[2]!=0?negif(lev[2]<0,(1<<shift)-1>>1):0); \
    new[3]=(lev[3]<<shift)+(lev[3]!=0?negif(lev[3]<0,(1<<shift)-1>>1):0);
#define RevOntLPF(new, lev, shift) \
    new[0]=(lev[0]<<shift)+((l<<shift)-l>>l); \
new[1]=(lev[1]<<shift)+((l<<shift)-l>>l); \
    new[2] = (lev[2] << shift) + ((1 << shift) -1 >> 1); \
```

```
Engineering: KlicsCode: CompPict: KlicsDec.c
    new[3]=ftev[3]<<shift)+((1<<shift.-1>>1):
/ Read a difference block and update memory */
*define DoXferDelta(addr.old.new.lev.dst.shift.mode.oct.nmode.buf) \
    HuffDecLev(lev,buf); \
    RevQntDelta(new,old,lev.shift) \
    PutData(addr.new.dst); \
    mode(cct)=oct==0?M_STOP:nmode:
/ * Read a block and update memory */
#define DoXfer(addr.new,lev,dst.shift.mcde.oct.nmode.buf) \
    HuffDecLev(lev.buf); \
    RevQnt(new,lev,shift) \
    PutData(addr,new.dst); \
   mode(oct)=oct==0?M_STOP:nmode;
/* Function Name: IntDecode
 Description:
                   Read a integer from bit file
   Arguments: bits - bits/integer now signed
   Returns:
               integer value
short
       IntDecode(short bits, Buf buf)
           i. lev=0. mask=1;
    int
    Boolean sign:
    /* Hardware compatable version */
   buf_rinc(buf);
    sign=buf_get(buf);
    for(i=0;i<bits-1;i++) (</pre>
       buf_rinc(buf);
       if (buf_get(buf)) lev |= mask;
       mask <<= 1;
    if (sign) lev= -lev;
   return(lev);
)
/* Function Name: HuffDecode
                   Read a Huffman coded integer from bit file
   Description:
   Returns:
              integer value
short
       HuffDecode (Buf buf)
   short lev=0, i;
   Boolean neg:
   /* Hardware compatable version */
   buf_rinc(buf);
   if (buf_get(buf)) {
       buf_rinc(buf);
       neg=buf_get(buf);
       ) ً ص
           buf_rinc(buf);
           lev++;
         while (lev<7 && !(buf_get(buf)));
       if (!(buf_get(buf))) (
           for(lev=0, i=0; i<7; i++) (
               lev<<=1;
              buf_rinc(buf);
```

```
Engineering: KlicsCode: CompPict: KlicsDec.c
```

```
if (buf_get(buf)) lev++;
        if (neg) lev= -lev;
    return(lev):
3
    Function Name: KlicsDChannel
    Description:
                     Decode a channel of image
    Arguments: dst - destination memory (and old for videos)
                 octs. Size - octaves of decomposition and image dimensions
                 normals - HVS weighted normals
                 lpf_bits - no of bits for LPF integer (image coding only)
 • /
void
        KlicsDecY(short *dst, int octs, int size(2), KlicsFrameHeader *frmh,
    KlicsSeqHeader *seqh, Buf buf)
    ine
             oct, mask, x, y, sub, step=2<<octs, blk(4), mode(4), base_mode=(frmh->
    Blk
             addr, new, old, lev;
    for(y=0;y<size(1);y+=step)</pre>
    for(x=0;x<size(0);x+=step)</pre>
    for (sub=0; sub<4; sub++) (
    mode(oct=octs-1)=base_mode;
    if (sub==0) mode(oct=octs-1) |= M_LPF;
    mask=2<<oct;
    do {
        GetAddr(addr,x,y,sub,oct,size,mask);
        switch(mode(oct)) (
        case M_VOID:
            GetData(addr,old,dst);
             if (BlkZero(old)) mode(oct)=M_STOP;
             else ( DoZero(addr,dst,mode,oct); }
             break;
        case M_SENDIM_STILL:
             buf_rinc(buf);
             if (buf_get(buf)) (
                 buf_rinc(buf);
if (buf_get(buf)) (
                     DoZero(addr,dst,mode,oct);
                 ) else
                     DoXfer(addr,new,lev.dst,frmh->quantizer(octs-oct),mode,oct,M_S
             ) else
                 mode [oct] =M_STOP:
            break:
        case M_SEND:
             buf_rinc(buf);
             if (buf_get(buf)) (
    buf_rinc(buf);
                 if (buf_get(buf)) (
    buf_rinc(buf);
                     if (buf_get(buf)) (
    GetData(addr,old,dst);
                          DoXferDelta(addr,old,new,lev,dst,frmh->quantizer(octs-oct)
                     } else (
                         DoZero(addr, dst, mode, oct);
                 ) else (
                     DoXfer(addr.new,lev,dst.frmh->quantizer(octs-oct),mode.oct,M_S
```

```
) else
                 mode(oct)=M_STOP:
             break;
        case M_STILL:
             buf_rinc(buf):
             if (buf_get(buf)) { Doxfer(addr,new,lev,dst,frmh->quantizer(octs-oct),:
             else mode(oct)=M_STOP:
             break:
        case H_LPFIM_STILL:
             IntDecLev(lev.seqh->precision-frmh->quantizer(0),buf);
             RevQntLPF(new,lev,frmh->quantizer(0));
             PutData(addr,new,dst):
             mode(oct)=M_QUIT;
             break;
        case M_LPFIM_SEND:
             buf_rinc(buf);
             if (buf_get(buf)) (
    GetData(addr,old.dst);
                  HuffDecLev(lev,buf);
                  RevQntDelta(new.old.lev.frmh->quantizer(0));
                  PutData(addr,new,dst);
             mode(oct)=M_QUIT:
             break;
        switch(mode(oct)) (
        case M_STOP:
             StopCounters(mode.oct.mask.blk.x.y.octs);
             break:
        case M_QUIT:
             break;
        default:
             DownCounters(mode.oct.mask.blk):
             break;
    ) while (mode(oct)!=M_QUIT);
)
    KlicsDecUV(short *dst, int octs, int size(2), KlicsFrameHeader *frmh, KlicsSeqHeader *seqh, Buf buf)
void
             oct. mask, x, y, X, Y, sub, step=4<<octs, blk{4}, mode{4}. base_mode=t addr, new, old, lev;
    int
    Blk
    for(Y=0;Y<size(1);Y+=step)
    for(X=0;X<size(0);X+=step)
    for(y=Y;y<size(1) && y<Y+step;y+=step>>1)
for(x=X;x<size(0) && x<X+step;x+=step>>1)
    for(sub=0;sub<4;sub++) (
    mode(oct=octs-1)=base_mode;
    if (sub==0) mode(oct=octs-1] |= M_LPF;
    mask=2<<oct;
    do (
        GetAddr(addr,x,y,sub,oct.size,mask);
        switch(mode(oct)) (
        case M_VOID:
             GetData(addr,old,dst);
             if (BlkZero(old)) mode(oct)=M_STOP;
             else ( DoZero(addr,dst,mode,oct); )
             break;
        case M_SENDIM_STILL:
```

```
Engineering:KlicsCode:CompPict:KlicsDec.c
```

```
but_rinc(but);
        if (buf_get(buf)) (
   buf_rinc(buf);
             if (buf_get(buf)) (
                 DoZero(addr.dst.mcde.oct);
             ) else (
                 DoXfer(addr,new.lev,dst.frmi->quantizer(octs-cct),mode.oct.M_S
        ) eise
            mode(oct)=M_STOP;
        break:
    case M_SEND:
        buf_rinc(buf);
        if (buf_get(buf)) (
             buf_rinc(buf);
             if (buf_get(buf)) (
                 buf_rinc(buf);
                 if (buf_get(buf)) (
                     GetData(addr,old,dst);
                     DoXferDelta(addr,old,new.lev,dst,frmh->quantizer(octs-oct)
                 ) else (
                     DoZero(addr, dst, mode, oct);
             } else {
                 DoXfer(addr.new.lev.dst.frmh->quantizer(octs-oct),mode,oct,M_S
        ) else
            mode(oct)=M_STOP;
        break;
    case M_STILL:
        buf_rinc(buf);
        if (buf_get(buf)) { DoXfer(addr,new,lev,dst,frmh->quantizer(octs-oct);;
        else mode(oct)=M_STOP;
        break;
    case H_LPFIM_STILL:
        IntDecLev(lev, seqh->precision-frmh->quantizer(0), buf);
        RevOntLPF(new, lev, frmh->quantizer(0));
        PutData(addr.new.dst);
        mode(oct)=M_QUIT;
        break;
    case M_LPF|M_SEND:
        buf_rinc(buf);
        if (buf_get(buf)) {
            GetData(addr.old.dst);
            HuffDecLev(lev.buf);
            RevQntDelta(new,old,lev,frmh->quantizer(0)):
            PutData(addr,new,dst);
        mode(oct)=M_QUIT;
        break;
    switch(mode(oct)) (
    case M_STOP:
        StopCounters(mode,oct,mask,blk,x,y,octs);
        break:
    case M_QUIT:
        break;
    default:
        DownCounters (mode, oct. mask, blk);
        break;
) while (mode(oct)!=M_QUIT);
```

```
Engineering:KlicsCode:CompPict:KlicsDec.c
 }
     Function Name: KlicsDecode
     Description:
                       Decode a frame to YUV (de)transformed image
     Arguments: src - destination result
                  dst - transformed destination memory (and old for videos)
     Returns:
                  whether this frame was skipped
                  KLCOPY(short 'dst, short 'src, long area);
KLHALF(short 'dst, short 'src, long size_0, long size_1);
 extern void
 extern void
 extern
                  KLICS3D2SEND(short *dst. long size_x, long size_y, short norms(4){
         void
 extern void
                  KLICS2D1STILL(short *dst. long size_x, long size_y, long lpfbits. KLICS3D2STILL(short *dst. long size_x, long size_y, long lpfbits.
extern void extern void
                  KLICS2D1SEND(short *dst, long size_x, long size_y, short norms(4){
#define flag_tree
                       0x1
*define flag_wave
                       0x2
void
         KlicsDecode(short *src[3], short *dst[3], KlicsSeqHeader *seqh,KlicsPrameH
             channel, i:
     short
             norms[4][2];
     unsigned long syncl, sync2;
     for(i=0;i<4;i++) (
         norms(i)(0)=(1<<frmh->quantizer(i)-1)-1;
         norms(i)[1]=frmh->quantizer(i);
     buf_rinit(buf);
     if (0!=(flags&flag_tree)) (
         syncl=GetTimerValue(&syncl);
         for(channel=0;channel<seqh->channels;channel++) {
                      size(2)=(seqh->sequence_size(0)>>(channel==0?0:seqh->sub_samp)
                          seqh->sequence_size(1)>>(channel==0?0:seqh->sub_sample(1))
                      tree_size[2]=(size[0]>>scale[0],size[1]>>scale[0]),
                      octs=seqh->octaves(channel==0?0:1);
*ifdef HQ
             if (0!=(frmh->flags&KFH_INTRA))
                 KLZERO(dst[channel],tree_size(0)*tree_size(1]);
             RLICSDCHANNEL(dst(channel),octs-1,tree_size[0],tree_size[1],(long)(seq
             if (channel==0) KlicsDecY(dst[channel].octs.tree_size.frmh.seqh.buf);
             else KlicsDecUV(dst[channel].octs,tree_size.frmh.seqh.buf);
#else
                      sub_tab[15]=(4,2,10,2+8*tree_size[0],10+8*tree_size[0],
             long
                          4*tree_size(0),2*tree_size(0],8+2*tree_size(0),10*tree_siz
4+4*tree_size(0),2+2*tree_size(0],10+2*tree_size(0),2+10*t
             if (0!=(frmh->flags&KFH_INTRA)) (
                 KLZERO(dst[channel],tree_size(0)*tree_size(1));
                 if (octs==3)
                      RLICS3D2STILL(dst(channel),tree_size(0),tree_size(1),(long)(se
                 else
                     KLICS2D1STILL(dst(channel), tree_size(0), tree_size(1), (long)(se-
             ) else
                 if (octs==3)
                     RLICS3D2SEND(dst(channel), tree_size(0), tree_size(1), anorms, abu
                 else
                     KLICS2D1SEND(dst(channel),tree_size(0),tree_size(1),&norms.&bu
*endif
        sync2=GetTimerValue(&sync2);
```

}

```
Engineering: KlicsCode: CompPict: KlicsDec.c
```

```
*tree#sync2-sync1;
if (0!=(flags&flag_wave)) (
    syncl=GetTimerValue(&syncl);
    for(channel=0;channel<seqn->channels:channel++) (
                 size(2)=(seqh->sequence_size(0)>>(channel==0?0:seqh->sub_sampl
        int
                     seqh->sequence_size[1]>>(channel==0?0:seqh->sub_sample[1])
                 wave_size(2) = (size(0) >> scale(1), size(1) >> scale(1)),
                 octs=seqh->octaves(channel==070:1);
        switch(segh->wavelet) (
        case WT_Haar:
if (scale(1)>scale(0))
                 KLHALF(dst[channel], src[channel], wave_size(0], wave_size(1]);
            else
                KLCOPY(dst[channel], src[channel], wave_size(0)*wave_size(1));
            HaarBackward(src(channel), wave_size, octs-scale(1));
            break;
        case WT_Daub4:
            if (scale(0)==0) (
                 if (scale[1]>scale[0])
                     KLHALF(dst(channel).src(channel), wave_size(0), wave_size(1)
                     KLCOPY(dst(channel), src(channel), wave_size(0)*wave_size(1)
                 Daub4Backward(src(channel), wave_size.occs-scale(1));
            ) else
                 if (channel==0) (
                     KLCOPY(dst(channel), src(channel), wave_size(0)*wave_size(1)
                     Backward3511(src(channel), wave_size, octs-scale(1));
                 ) else
                     TOPBWD(dst(channel), src(channel), wave_size(0), wave_size(1)
            break;
    sync2=GetTimerValue(&sync2);
    *wave=sync2-sync1;
1
```

Engineering:KlicsCode:CompPict:KlicsCodec.c

```
© Copyright 1993 KLICS Limited
    All rights reserved.
    Written by: Adrian Lewis

    Klics Codec

*include 'ImageCodec.h'
*include <FixMath.h>
*include <Errors.h>
*include <Packages.h>
*ifdef PERFORMANCE
    #include <Perf.h>
    extern TP2PerfGlobals ThePGlobals:
*endif
#1fdef DEBUG
    #define DebugMsg(val) DebugStr(val)
    *define DebugMsg(val)
*endif
#define WT_Haar 0
*define WT_Daub4 1
#define None
*define Use8
                1
#define Usel6
#define Use32
#define UseF32
/* Version information */
#define KLICS_CODEC_REV
*define codecInterfaceVersion
                                  /* high word returned in component GetVersion
#define klicsCodecFormatName
                                'Klics'
*define klicsCodecFormatType
                                'klic'
pascal ComponentResult
KlicsCodec(ComponentParameters *params,char **storage);
pascal ComponentResult
KLOpenCodec(ComponentInstance self);
pascal ComponentResult
KLCloseCodec(Handle storage,ComponentInstance self);
pascal ComponentResult
KLCanDoSelector(short selector);
pascal ComponentResult
KLGetVersion();
pascal ComponentResult
MUGetCodecInfo(Handle storage.CodecInfo *info):
```

```
Engineering:KlicsCode:CompPict:KlicsCodec.c
pascal ComponentResult
KLGetMaxCompressionSize(Handle storage,PixMapHandle src,const Rect *srcRect,short -
    CodecQ quality,long *size);
pascal ComponentResult
KLGetCompressedImageSize(Handle storage, ImageDescriptionHandle desc, Ptr data, long -
    DataProcRecordPtr dataProc.long *size);
pascal ComponentResult
KLPreCompress(Handle storage,register CodecCompressParams *p);
pascal long
KLPreDecompress(Handle storage, register CodecDecompressParams *p);
pascal long
KLBandDecompress(Handle storage, register CodecDecompressParams *p);
pascal long
KLBandCompress(Handle storage,register CodecCompressParams *p);
pascal ComponentResult
KLGetCompressionTime(Handle storage, PixMapHandle src,const Rect *srcRect, short dep
        CodecQ *spatialQuality,CodecQ *temporalQuality,unsigned long *time);
/* Function:
               KlicsCodec
    Description: KlicsCodec main despatcher
#ifdef DECODER
pascal ComponentResult
KlicsDecoder(ComponentParameters *params, char **storage)
felse
#ifdef ENCODER
pascal ComponentResult
RlicsEncoder(ComponentParameters *params,char **storage)
telse
pascal ComponentResult
KlicsCodec(ComponentParameters *params,char **storage)
#endif
sendif
   OSETT
          err:
    switch ( params->what ) {
   case kComponentOpenSelect:
       err=CallComponentFunction(params,(ComponentFunction) KLOpenCodec); break;
           kComponentCloseSelect:
   case
       err=CallComponentPunctionWithStorage(storage,params,(ComponentFunction)KLC
           kComponentCanDoSelect:
        err=CallComponentFunction(params,(ComponentFunction)KLCanDoSelector); brea
   case kComponentVersionSelect :
       err=CallComponentFunction(params.(ComponentFunction)KLGetVersion); break;
#ifdef DECODER
   case codecPreCompress:
   case codecBandCompress:
       err=codecUnimpErr; break;
```

case codecPreCompress:

Engineering: KlicsCode: CompPict: KlicsCodec.c

```
err=CaFTComponentFunctionWithStorage(scorage.params, (ComponentFunction)KLP
   case codecBandCompress:
        err=CallComponentFunctionWithStorage(storage.params.(ComponentFunction)KLB
rendif
#1fdef ENCODER
    case codecPreDecompress:
    case codecBandDecompress:
        err=codecUnimpErr: break;
*else
    case codecPreDecompress:
        err=CallComponentFunctionWithStorage(storage,params.(ComponentFunction)KLP
    case codecBandDecompress:
        err=CallComponentFunctionWithStorage(storage,params,(ComponentFunction)KLB
*endif
    case codecCDSequenceBusy:
                                          /* our codec is never asynchronously busy
        err=0; break;
    case codecGetCodecInfo:
        err=CallComponentFunctionWithStorage(storage,params,(ComponentFunction)KLG
    case codecGetCompressedImageSize:
        err=CallComponentFunctionwithStorage(storage,params.(ComponentFunction)KLG
    case codecGetMaxCompressionSize:
        err=CallComponentFunctionWithStorage(storage,params,(ComponentFunction)KLG
    case codecGetCompressionTime:
        err=CallComponentFunctionWithStorage(storage,params,(ComponentFunction)KLG
    case codecGetSimilarity:
        err=codecUnimpErr; break;
    case codecTrimLmage:
        err=codecUnimpErr; break;
    default:
        err=paramErr; break;
    if (err!=noErr)
        DebugMsg(*\pCodec Error*);
    return(err);
#include <Memory.h>
#include <Resources.h>
#include <OSUtils.h>
#include <SysEqu.h>
#include <StdIO.h>
#include <Time.h>
#include <Strings.h>
#include <String.h>
#include 'Bits3.h'
#include *KlicsHeader.h*
#include *KlicsEncode.h*
        DebugString(char *string)
void
    DebugStr(string);
```

Engineering: KlicsCode: CompPict: KlicsCodec. C

```
extern short gResRef;
typedef struct ( **info:
     CodecInfo
            cab(4):
use(4):
     short
) SharedGlobals:
typedef struct (
                                              /* Encoding parameters */
/* YUV Frame buffer */
     KlicsERec
                    kle:
               *src(3);
     short
                                              /* YUV Frame buffer */
               *dst(3):
     short
                                              /* Encoded pixmap data */
/* Size of Previous Frame Buffer */
     Ptr
               pixmap;
     long
               size;
                                               /* Which lookup table are we using for colour /* Tree. Wave. Out scales 0=Original. -l=Doubl
     long.
               using:
     long
               scale(3):
     unsigned long prev_frame;
unsigned long real_frame;
                                              /* Previous frame number */
                                               /* Previous real frame (no skips) */
                                               /* Previous displayed frame */
     unsigned long dpy_frame;
                                              /* First frame in play sequence */
     unsigned long run_frame:
                                               /* System overhead for previous frame */
     unsigned long sys_time; unsigned long tree_time;
                                               /* Typical tree decode time (not skip) */
                                               /* Typical wavelet transform time */
/* Typical display time */
     unsigned long wave_time;
     unsigned long dpy_time;
                                               /* Time of first run frame */
     unsigned long run_time;
                                               /* Time at last key frame */
     unsigned long key_time;
     unsigned long _,
Boolean out[15];
-delobals *sharedGlob;
     unsigned long sync_time;
                                               /* Sync time */
                                               /* Displayed? */
| Globals;
/* Scaling scenarios: Tree Wave Out
        1 0: Internal calculations are Quarter size, output Original size (interpo
1 1: Internal calculations are Quarter size, output Quarter size
        1 1: Internal calculations are Original size, output Quarter size 0 0: Internal calculations are Original size, output Original size
         0 -1: Internal calculations are Original size, output Double size
          KLDeallocate(Globals **glob);
void
 /* Klics Function Definitions */
extern int    KlicsEncode(short *src[3], short *dst[3], KlicsE kle);
extern Boolean KlicsDecode(short *src[3], short *dst[3], KlicsSeqHeader *seqh,Kli
long mode, long scale[3], unsigned long *tree, unsigned long *wave);
 /********************
     Memory allocation/deallocation routines
 OSErr
 MemoryError()
      OSErr theErr:
 #ifdef DEBUG
      if (0!=(theErr=MemError()))
```

DebugStr' " \ pMemoryError");

```
Engineering: KlicsCode: CompPict: KlicsCodec.c
+endif
    return(theErr);
CSErr
FreePtr(Ptr 'ptr)
    CSErr theErr=0:
    if (*per:=nil) (
        DisposePtr(*ptr):
        *ptr=nil:
        theErr=MemoryError();
    return(theErr);
}
#define FreePointer(handle,err) \
    if (noErr!=(err=FreePtr!(Ptr*)(&handle)))) return(err)
                Colour8(Ptr *);
Colour16(Ptr *);
extern OSErr
extern OSErr
extern OSErr
                UV32Table(Ptr *):
               RGBTable(Ptr *);
extern OSErr
KLGetTab(Globals **glob,long new)
    OSErr theErr=0;
SharedGlobals *sGlob=(*glob)->sharedGlob:
long old=(*glob)->using;
    if (old!=new) {
        if (old!=None) (
            sGlob->use(old-1)--;
             if (sGlob->use(old-1)==0) (
                 FreePointer(sGlob->tab(old-1),theErr);
        }
        if (new!=None) (
             if (sGlob->use(new-1)==0)
                 switch(new) (
*ifndef ENCODER
                 case Use8:
                     if (noErr!=(theErr=Colour8(&sGlob->tab(new-1))))
                         return(theErr):
                     break:
                 case Use15:
                     if (noErr!=(theErr=Colour16(&sGlob->tab(new-1))))
                         recurn(theErr);
                     break:
                 case Use32:
                     if (noErr!=(theErr=UV32Table(&sGlob->tab(new-1))))
                         return(theErr);
                     break:
*endif
*ifndef DECODER
                 case UseF32:
                     if (noErr!=(theErr=RGBTable(&sGlob->tab(new-1));)
                         return(theErr):
                     break:
```

```
Engineering:KlicsCode:CompPict:KlicsCodec.c
•endif
             (*glob) ->usinginew;
             sGlob->use(new-1)++;
        )
    return(theErr):
OSErr
KLFree(Globals **glob)
    OSErr
            theErr=0:
    FreePointer((*glob)->src(0),theErr);
    FreePointer((*glob)->dst(0),theErr;;
    FreePointer((*glob)->pixmap,theErr);
    (*glob) ->size=0:
    return(theErr);
*define NewPointer(ptr,type,size) \
    saveZone=GetZone(); \
    SetZone(SystemZone()); \
    if (nil==(ptr=(type)NewPtr(size))) ( \
        Set2one(ApplicZone()); \
        if (nil==(ptr=(type)NewPtr(size))) ( \
             SetZone(saveZone); \
             return(MemoryError()); \
        ) \
    SetZone(saveZone);
ComponentResult
KLMalloc(Globals **glob, short height, short width, long pixelSize)
٠ (
    long
            ysize, uvsize;
    THZ
             saveZone;
    ysize= (long)height * (long)width * (long)sizeof(short);
    uvsize = ysize>>2;
    if ((*glob)->size != ysize) (
        KLFree(glob);
        (*glob) -> size = ysize:
       ('glob)->prev_frame=-1; /' frame doesn't contain valid data '/
        /* Keep Src and Dst separate because of their large sizes */
        ysize=(long)height * (long)width * (long)sizeof(short) >> 2*(*glob)->scale
        uvsize = ysize>>2;
        NewPointer(('glob) ->src(0), short *, ysize+uvsize+uvsize+16);
('glob) ->src(1) = (short *)(((long)('glob) ->src(0) + ysize + 3L) & 0xFFFFF
('glob) ->src(2) = (short *)(((long)('glob) ->src(1) + uvsize + 3L) & 0xFFFFF.
        ysize=(long)height * (long)width * (long)sizeof(short) >> 2*(*glob)->scale
        uvsize = ysize>>2;
        NewPointer((*glob)->dst[0], short *, ysize+uvsize+uvsize+16);
```

```
Engineering:KlicsCode:CompPict:KlicsCodec.c
         NewPointer((*glob)->pixmap.Ptr.pixelSize/6*height*width<<li);</pre>
     return(noErr);
CSErr
 RescurcePrror()
     CSErr
             theErr:
 *ifdef DEBUG
     if (0!=(theErr=ResError()))
         DebugStr('\pResourceError');
 ≢endif
     return(theErr);
 #ifdef COMPONENT
     #define ResErr(resfile.err) \
         if (0!=(err=ResourceError())) ( \
             if (resfile!=0) CloseComponentResFile(resfile): \
             return(err); \
#else
     #define ResErr(resfile.err) \
         if (0!=(err=ResourceError())) ( \
             return(err); \
-endif
ComponentResult
KLOpenInfoRes(ComponentInstance self, Handle *info)
*pragma unused(self)
    short
            resFile=0;
    OSETT
            theErr=noErr;
   'if (*info) (
        DisposHandle(*info);
        *info=nil;
Fifdet COMPONENT
    resFile=OpenComponentResFile((Component)self);
    ResErr(resFile, theErr);
*else
   UseResFile(gResRef);
=endif
    *info=GetlResource(codecInfoResourceType, 128);
    *info=GetlResource(codecInfoResourceType, 129);
    ResErr(resFile, theErr);
    LoadResource ( * info);
    ResErr(resFile, theErr);
    DetachResource(*info);
wifdef COMPONENT
   CloseComponentResFile(resFile);
#endif
    return(theErr);
pascal ComponentResult
KLOpenCodec (ComponentInstance self)
   Globals
                    **glob:
```

```
Engineering:KlicsCode:CompPict:KlicsCodec.c
                    •sGlob:
     SharedGlobals
     THZ
                     saveZone:
     Soolean
                     inAppHeap:
    ComponentResult result = noErr: -
     short
           resFile=CurResFile():
    DebugMsg(*\pOpen Codec - begin*);
    rf ( (glob = (Globals **)NewHandleClear(sizeof(Globals);) == nil ) (
        return(MemoryError());
     } else HNoPurge((Handle)glob);
    SetComponentInstanceStorage(self,(Handle)glob);
    saveZone = GetZone();
    inAppHeap = ( GetComponentInstanceA5(self) != 0 );
    if (!inAppHeap)
        SetZone(SystemZone());
    if ( (sGlob=(SharedGlobals*)GetComponentRefcon((Component)self)) == nil ) !
        if ( (sGlob = (SharedGlobals*)NewPtrClear(sizeof(SharedGlobals))) == nil )
            result = MemoryError();
            goto obail:
        SetComponentRefcon((Component)self,(long)sGlob);
    (*glob)->sharedGlob = sGlob;
                                     // keep this around where it's easy to get at
    if ( sGlob->info == mil || *(Handle)sGlob->info == mil ) (
        result=KLOpenInfoRes(self,&(Handle)(sGlob->info));
        HNoPurge ( (Handle) sGlob->info);
obail:
    SetZone(saveZone);
    if ( result != noErr && sGlob != nil ) (
   if ( sGlob->info )
            DisposHandle((Handle)sGlob->info);
        DisposPtr((Ptr)sGlob);
        SetComponentRefcon((Component)self,(long)nil);
    (*glob)->size=0;
    DebugMsg(*\pOpen Codec - end*);
   recurn(result);
pascal ComponentResult
KLCloseCodec(Handle storage, ComponentInstance self)
   SharedGlobals
                   *sGlob:
   Globals
                   **glob = (Globals **)storage;
   DebugMsg(*\pClose Codec - begin*);
   HLock (storage);
   if (glob) {
       KLPree (glob) :
       KLGetTab(glob, None);
        if (CountComponentInstances((Component)self) == 1) (
            if ( (sGlob=(SharedGlobals=)(*glob)->sharedGlob) != nil ) (
                if ( sGlob->info )
    HPurge((Handle)sGlob->info);
       DisposHandle((Handle)glob);
```

```
Engineering: KlicsCode: CompPict: KlicsCodec.c
        height = 120-
    if (time)
        *time = (width * height * 11);
    if (spatialQuality && *spatialQuality==codecLosslessQuality)
        *spacialQuality = codecMaxQuality;
    if :temporalQuality && *temporalQuality==codecLosslessQuality)
        *temporalQuality = codecMaxQuality;
    return(noErr):
ï
 * Extends dimensions to make a multiples of 32x16
#define KLExtendWidth(dim) 31-(dim-1&31)
#define KLExtendHeight(dim) 15-(dim-1&15)
pascal ComponentResult
KLGetMaxCompressionSize(Handle storage, PixHapHandle src, const Rect *srcRect, short -
    CodecQ quality, long 'size)
*pragma unused(storage.src.depth.quality)
    short width = srcRect->right - srcRect->left:
    snort height = srcRect->bottom - srcRect->top:
    /* test by just doing RGB storage */
    *size * 3 * (width+KLExtendWidth(width)) * (height+KLExtendHeight(height));
    return(noErr);
pascal ComponentResult
KLGetCompressedImageSize(Handle storage,ImageDescriptionHandle desc.Ptr data,long
    DataProcRecordPtr dataProc, long *size)
*pragma unused(storage.dataSize,dataProc.desc)
    short
          frmh_size:
    long
           data_size;
    if ( size == nil ) (
       return(paramErr):
    irmh_size=((KlicsHeader *)data)->description_length; '
    data_size=((KlicsFrameHeader *)data)->length;
    *size=(long)frmh_size+data_size:
    return(noErr);
1
void
       KLSetup(Boolean still, short width, short height, CodecQ space, CodecQ tem
   kle->seqh.head.description_length=sizeof(RlicsSeqHeader);
    kle->seqh.head.version_number(0)=0;
   kle->seqh.head.version_number[1]=1;
    kle->seqh.sequence_size(0)=width;
    kle->seqh.sequence_size(1)=height;
    kle->seqn.sequence_size(2)=0;
   kle->seqh.sub_sample(0)=1;
   kle->seqh.sub_sample[1]=1;
   kle->seqh.wavelet=WT_Daub4;
```

(*glob)->scale[0]=0; (*glob)->scale[1]=0;

```
Engineering:KlicsCode:CompPict:KlicsCodec.c
    kle->seqh.precTsion=10;
    kle->seqh.cctaves(0)=3;
    kle->seqh.octaves(1)=2;
    <le->frmh.head.description_length=sizeof(KlicsFrameHeader);
    kle->frmh.head.version_number(0)=0;
    kle->frmh.head.version_number[1]=1;
    kle->encd.bpf_in=(2133+temp*160)/8;
                                                /* High = 64000 bits/frame, Poor = 1
    kle->encd.opf_out=kle->encd.opf_in;
    kle->encd.buf_size=kle->encd.bpf_in*4;
    kle->encd.quant=16-(space*15)/1023;
    xle->encd.thresh=1.0;
    kle->encd.compare=1.0;
    kle->encd.base(0)=0.10;
    kle->encd.base(1)=0.10;
    kle->encd.base(2)=0.20;
    kle->encd.base(3)=0.50;
    kle->encd.base[4]=1.00;
    kle->encd.intra=still;
    kle->encd.auto_q=true;
   kle->encd.buf_sw=true:
    kle->encd.prevquact=1;
    kle->encd.prevbytes=13;
#ifndef DECODER
pascal ComponentResult
KLPreCompress(Handle storage, register CodecCompressParams *p)
   ComponentResult
                        result;
   CodecCapabilities
                        *capabilities = p->capabilities:
   short
                        width=(*p->imageDescription)->width+(capabilities->extendW
                        height=(*p->imageDescription)->height+(capabilities->exten-
**glob=(Globals **)storage;
   short
   Globals
   KlicsE
                        kle=&(*glob)->kle;
   Handle
                        ext=NewHandle(sizeof(KlicsSeqHeader));
   DebugHsg(*\pKLPreCompress*);
   HLock(storage);
   if (MemError()!=noErr) return(MemError());
   switch ( (*p->imageDescription)->depth )
       case 24:
           capabilities->wantedPixelSize = 32;
           kle->seqh.channels=3;
           if (noErr!=(result=KLGecTab(glob.UseF32)))
               return(result);
           break;
       default:
           return(codecConditionErr);
           break;
   /* Going to use 3 octaves for Y and 2 for UV so the image must be a multiple o
   capabilities->bandMin = height;
   capabilities->bandInc = capabilities->bandMin;
   capabilities->flags=codecCanCopyPrevComp(codecCanCopyPrev;
```

```
Engineering:KlicsCode:CompPict:KlicsCodec.c
    (*glob) ->scale(2) =0;
    if (noErr!=(result=KLMalloc(glob,height.width.0))) return result;
    KLSetup(p->sequenceID==0, width, height, (*p->imageDescription) ->spatialQuality, (
    BlcckMove((Ptr)&kle->seqh.*ext,sizeof(KlicsSeqHeader)):
    if (ncErr!=!result=SetImageDescriptionExtension(p->imageDescription.ext,xlicsC
    return result:
    HUnlock(storage);
    DebugMsg(*\pKLPreCompress success*);
    recurn(result):
*endif
*ifndef ENCODER
pascal long
KLPreDecompress(Handle storage, register CodecDecompressParams *p)
    ComponentResult
                         result:
    CodecCapabilities
                         *capabilities = p->capabilities:
    Rect
                         dRect = p->srcRect:
    long
                         width:
    long
                         height:
    long
                         charnels:
    Globals
                         **glob=(Globals **)storage;
    KlicsE
                         kle;
    Handle
                         ext;
    OSErr
                erri
    DebugMsg(*\pKLPreDecompress*);
    if ( :TransformRect(p->matrix,&dRect,nil) )
    return(codecConditionErr);
    HLock(storage);
    kle=&(*glob)->kle:
    switch ( (*p->imageDescription)->depth ) {
        case 24:
            switch(p->dstPixMap.pixelSize) (
            case 32:
                capabilities->wantedPixelSize = 32;
                if (p->conditionFlags&codecConditionNewDepth) {
                     if (noErr!=(err=KLGetTab(glob,Use32)))
                        return(err):
                break:
            case 16:
                capabilities->wantedPixelSize = 16;
                if (p->conditionFlags&codecConditionNewDepth) (
                    if (noErr!=(err=RLGecTab(glob.Usel6)))
                        return(err);
                break:
            case 8:
                capabilities->wantedPixelSize = 8;
                if (p->conditionFlags&codecConditionNewClut) (
                    if (noErr!=(err=KLGetTab(glob, Use8)))
                        return(err);
                break;
            channels=3:
            break:
```

```
Engineering:KlicsCode:CompPict:KlicsCodec.c
        dafault:
            return(codecConditionErr);
            break:
   }
   if (noErr!=(result=GetImageDescriptionExtension(p->imageDescription, &ext, klics-
   BlockMove(*ext,(Ptr)&kle->seqh,sizeof(KlicsSeqHeader));
   if (channels==1) kle->seqh.channels=1;
   /* Going to use 3 octaves for Y and 2 for UV so the image must be a multiple o
*ifdef HQ
   (*glob) ->scale(0]=0; /* Tree scale */
#else
   (*glob)->scale[0]=1; /* Tree scale */
*endif
   width=kle->seqh.sequence_size(0);
   height=kle->seqh.sequence_size(1);
   switch((*glob)->scale(0)) (
   case 1: /* Quarter size internal */
        (*glob)->scale(1)=1;
        if (p->matrix->matrix(0)(0)==p->matrix->matrix(1)(1))
            switch(p->matrix->matrix(0)(0)) (
            case 32768:
                capabilities->flags=codecCanScale;
                capabilities->extendWidth=width/2-dRect.right;
                Capabilities->extendHeight=height/2-dRect.bottom;
                (*glob) ->scale(2)=1;
               break;
            case 65536:
                capabilities->extendWidth=width-dRect.right;
                capabilities->extendHeight=height-dRect.bottom;
                (*glob) ->scale[2]=0;
                break;
           default:
                capabilities->extendWidth=0;
                capabilities->extendHeight=0;
                (*glob)->scale[2]=0;
               break;
       else (
           capabilities->extendWidth=0;
           capabilities->extendHeight=0;
           (*glob) ->scale(2)=0:
   break; case 0: /* Full size internal */
       if (p->matrix->matrix(0)(0)==p->matrix->matrix(1)(1))
           switch(p->matrix->matrix(0)(0)) {
case 32768:
               capabilities->flags=codecCanScale;
               capabilities->extendWidth=width/2-dRect.right;
               capabilities -> extendHeight=height/2-dRect.bottom;
               (*glob) ->scale(1)=1;
               (*glob) ->scale(2)=1;
               break:
           case 131072:
               capabilities->flags=codecCanScale;
               capabilities->extendWidth=width*2-dRect.right;
               capabilities->extendHeight=height*2-dRect.bottom;
               (*glob) ->scale[1]=0;
               (*glob) ->scale(2) =-1;
```

```
Engineering: KlicsCode: CompFict: KlicsCodec.c
                      break;
                case 65536:
                      capabilities->extendWidth=width-dRect.right:
                      capabilities->extendHeight=height-dRect.bottcm;
                      (*glob) ->scale[1]=0;
                      (*glob)->scale(2)=0;
                      oreak;
                default:
                      capabilities->extendWidth=0;
                      capabilities->extendHeight=0;
                      (*glob) ->scale(1)=C:
                      (*glob) ->scale(2)=0;
           else (
                capabilities->extendWidth=0;
                capabilities->extendHeight=0;
                (*glob) ->scale(1)=0;
                (*glob)->scale(2)=0;
           break:
     }
     capabilities->bandMin = height:
     capabilities->bandInc = capabilities->bandMin;
     capabilities->flags!=codecCanCopyPrev!codecCanCopyPrevComp!codecCanRemapColor;
     if (noErr!=(result=KLMalloc(glob,height,width,capabilities->wantedPixelSize)))
     HUnlock(storage);
     DebugMsg('\pKLPreDecompress success');
     return(result):
*endif
/* Test Versions in C - Colour.c */
          RGB2YUV32(long *pixmap, short *Yc, short *Uc, short *Vc, int area, int wid YUV2RGB32(long *pixmap, short *Yc, short *Uc, short *Vc, int area, int wid YUV2RGB32x2(Ptr table,long *pixmap, short *Yc, short *Uc, short *Vc, int a
void
void
void
/* Assembler versions - Colour.a */
OUT32X2(Ptr table,long *pixmap,short *Y,short *U,short *V,long width,long height,1
OUT32X2D(Ptr table,long *pixmap,short *Y,short *U,short *V,long width,long height,
OUT32 (Ptr table, long *pixmap, short *Y, short *U, short *V, long width, long height, lon-
OUT32D(Ptr table,long *pixmap,short *Y,short *U,short *V,long width,long height,lo
OUT8X2(Ptr table,long *pixmap,short *Y,short *V,short *V.long width,long height,lo
OUTS(Ptr table.long *pixmap,short *Y,short *U,short *V,long width,long height,long
OUT16X2(Ptr table.long *pixmap, short *Y, short *U, short *V, long width.long height, l
OUT16(Ptr table,long *pixmap,short *Y,short *U,short *V.long width,long height,long IN32(Ptr table,long *pixmap,short *Y,short *U,short *V,long width,long height,long
/* Assembler versions - Color2.a */
          RGB2YUV2(long *pixmap, short *Yc, short *Uc, short *Vc, int area, int widt
YUV2RGB2(long *pixmap, short *Yc, short *Uc, short *Vc, int area, int widt
void
void
          YUV2RGB3(long *pixmap, short *Yc, short *Uc, short *Vc, int area, int widt)
void
          GREY2Y(long *pixmap, short *Yc, int area, int width, int cols);
Y2GREY(long *pixmap, short *Yc, int lines, int width, int cols);
Y2GGG(long *pixmap, short *Yc, int lines, int width, int cols);
void
void
void
/*YUV2RGB4((*glob)->Table.pixmap.src[0],src[1],src[2],cols*(*desc)->height>>scale.'
YUV2RGB5((*glob)->Table.pixmap.src[0],src[1],src[2],cols*(*desc)->height,width>>sc
*pragma parameter __DO MicroSeconds
```

```
Engineering:KlicsCode:CompPict:KlicsCodec.c
pascal unsigned long MicroSeconds(void) = (0x4EBO, 0x81E1, 0x64C);
unsigned long
                 GetTimerValue(unsigned long 'TimerRes)
     *TimerRes = CLOCKS_PER_SEC;
    return(MicroSeconds());
#ifndef DECODER
pascal long
KLBandCompress(Handle storage, register CodecCompressParams 'p)
*pragma unused(storage)
    Globals
                           **glob = (Globals **)storage;
    ImageDescription
                          **desc = p->imageDescription;
    char
                          *baseAddr:
    short
                          rowBytes;
    Rect
                          sRect:
    long
                          offsetH, offsetV;
    OSETT
                          result = noErr;
    short
                          *src(3), *dst(3);
    long
                          'pixmap;
    int
                          width=(*desc)->width+KLExtendWidth((*desc)->width);
    int
                          height=(*desc)->height+KLExtendHeight((*desc)->height);
    int
                          hwidth=width>>1, hheight=height>>1;
    int
                          bytes:
    KlicsE
                          kle:
    char
                          mmuMode=1;
    char
                          intra[]="\pENC:Intra-mode", inter[]="\pENC:Inter-mode";
    SharedGlobals
                          *sGlob;
fifdef PERFORMANCE
     (void) PerfControl (ThePGlobals, true);
#endif
    DebugMsg('\pBandCompress');
    HLock ((Handle)glob);
    kle=&(*glob)->kle;
    SGlob=(*glob)->sharedGlob;
    rowBytes = p->srcPixMap.rowBytes & 0x3fff;
    sRect = p->srcPixMap.bounds;
switch ( p->srcPixMap.pixelSize ) (
    case 32:
        offsetH = sRect.left<<2;
        break;
    case 16:
        offsetH = sRect.left<<1;
        break:
    case 8:
        offsetH = sRect.left;
        break:
    default:
        result = codecErr:
        DebugMsg("\pError");
        goto bail;
    offsetV = sRect.top * rowBytes;
    baseAddr = p->srcPixMap.baseAddr + offsetH + offsetV;
pixmap=(long *)baseAddr;
/* PSMakeFSSpec(0.0.*\pUser:crap001*,&fsspec);
FSpCreate(&fsspec.'????','????',-1);
```

```
Engineering: KlicsCode: CompPict: KlicsCodec.c
      FSpOpenDF(&fsspec.fsWrPerm,&fileRefNum);
      area=height rowBytes;
      FSWrite(fileRefNum.&area,(long*)pixmap);
      FSClose(fileRefNum); */
      src[0]=(*glob)->src[0]; src[1]=(*glob)->src[1]; src[2]=(*glob)->src[2];
dst[0]=(*glob)->dst[0]; dst[1]=(*glob)->dst[1]; dst[2]=(*glob)->dst[2];
      switch(kle->seqn.channels) (
      case 3:
          IN32(sClob->tab(UseF32-1),pixmap.src(0),src(1),src(2),width,height,rowByte
          break;
      }
          Klics encode
              #ifdef DEBUG
     if (p->callerFlags&codecFlagUseImageBuffer) DebugStr(*\pUseImageBuffer*);
     if (p->callerFlags&codecFlagUseScreenBuffer) DebugStr(*\pUseScreenBuffer*): /*
     if (p->callerFlags&codecFlagUpdatePrevious) DebugStr('\pUpdatePrevious');
if (p->callerFlags&codecFlagNoScreenUpdate) DebugStr('\pNoScreenUpdate');
     if (p->callerFlags&codecFlagDontOffscreen) DebugStr(*\pDontOffscreen*); /*
if (p->callerFlags&codecFlagUpdatePreviousComp) DebugStr(*\pUpdatePreviousComp)
                                                                                             /•
     if (p->callerFlags&codecFlagForceKeyFrame) DebugStr('\pForceKeyFrame');
     if (p->callerFlags&codecFlagOnlyScreenUpdate) DebugStr(*\pOnlyScreenUpdate*);
 *endif
     kle->buf.buf=(unsigned long *)(p->data+sizeof(KlicsFrameHeader));
kle->encd.intra=(p->temporalQuality==0);
     kle->frmh.frame_number=p->frameNumber;
     bytes=KlicsEncode(src, dst, kle);
     BlockMove((Ptr)&kle->frmh,p->data.sizeof(KlicsFrameHeader));
     bytes+=sizeof(KlicsFrameHeader);
     (*glob)->prev_frame=p->frameNumber:
    p->data+=bytes:
    p->bufferSize=bytes:
     (*p->imageDescription)->dataSize=bytes;
    p->similarity=(kle->encd.intra?0:Long2Fix(244));
     p->callerFlags=0;
/* p->callerFlags:=codecFlagUsedImageBuffer:(kle->encd.intra?codecFlagUsedNewImag
bail:
    HUnlock((Handle)glob);
#ifdef PERFORMANCE
    if(0!=(result=PerfDump(ThePGlobals, '\pEncode.perf',false,0)))
        return(result);
#endif
    DebugMsg(*\pBandCompress success*);
    return(result);
*endif
/ Display stuff for debugging
    CGrafPtr
                wPort, savePort;
```

```
Engineering:KlicsCode:CompPict:KlicsCodec.c
     Rect
                  rect;
     3tr255
                  str:
     GetPort((GrafPtr *)&savePort):
     GecCWMgrPort(EwPort);
     SetPort((GrafPtr)wPort);
    SetRect(&rect.0.0.50,30);
    ClipRect(&rect);
    EraseRect(&rect);
    NumToString(frmh->frame_number,str);
    MoveTo(0,20);
    DrawString(str);
    if (frmh->flags&KFH_INTRA) (
        SetRect(&rect, 0, 30, 50, 65);
        ClipRect(&rect);
        EraseRect (&rect);
        NumToString(frmh->frame_number/24,str);
        MoveTo (0,50);
        DrawString(str);
    SetRect(&rect.-2000,0.2000.2000);
    ClipRect(&rect);
    SetPort((GrafPtr)savePort); */
*define flag_tree
                     0x1
*define flag_wave
                     0x2
*define flag_show
                     0x4
*define flag_full
                     0×8
*define DURATION
                     65666
long
        ModeSwitch(Globals *glob, KlicsFrameHeader *frmh)
(
    lone
            mode=0, i, fps;
    Boolean repeat=glob->prev_frame==frmh->frame_number,
           next=glob->prev_frame+1==frmh->frame_number;
    CGrafPtr
                wPort, savePort;
    Rect
                rect;
    Str255
                Str:
   DebugMsg("\pModeSwitch - begin");
   if (frmh->frame_number==0)
       for(i=0;i<15;i++) glob->out[i]=false;
   if (repeat) (
       glob->run_time=0;
DebugMsg('\pModeSwitch - repeat (end)');
       return(flag_snow)flag_full);
   if (next)
       switch(frmh->flags) (
       case KFH_SKIP:
           DebugHsg(*\pModeSwitch - next/skip*);
           glob->prev_frame=frmh->frame_number;
if (glob->sys_time>DURATION) (
                glob->run_time=0;
                if (glob->real_frame!=glob->dpy_frame)
                    mode | = flag_wave | flag_show;
            } else (
                unsigned long frame, late;
                frame=glob->run_frame+(glob->sync_time-glob->run_time)/DURATION;
                late=(glob->sync_time-glob->run_time) %DURATION;
                if (frame<=glob->prev_frame && glob->real_frame!=glob->dpy_frame)
```

```
Engineering: KlicsCode:CompPict:KlicsCodec.c
             mode!=flag_wave!flag_show;
         if (frame<=glob->prev_frame && late+glob->wave_time+glob->dpy_time
             mode!=flag_wave flag_show: */
    break;
case KFH_INTRA:
    DebugMsg("\pMcdeSwitch - next/intra"):
    mode=flag_tree;
    glob->prev_frame=frmh->frame_number:
    glob->real_frame=glob->prev_frame:
     if (glob->sys_time>DURATION) (
         glob->run_time=0:
         mode:=flag_wave:flag_show:flag_full;
    ) else
         if (glob->run_time==0) (*/
             glob->key_time=glob->sync_time-glob->run_cime:
glob->run_time=glob->sync_time-glob->sys_time:
             glob->run_frame=glob->prev_frame:
             mode:=flag_wave;flag_show;flag_full;
         ) else (
             unsigned long frame, lace;
             frame=glob->run_frame+(glob->sync_time-glob->run_time)/DURATIO
             late=(glob->sync_time-glob->rur_time) *DURATION;
             if (frame<=glob->prev_frame)
                 mode | = flag_wave|flag_show|flag_full;
         3 . /
    break;
default:
    DebugMsg("\pNodeSwitch - next/inter");
    mode=flag_tree;
    glob->prev_frame=frmh->frame_number;
glob->real_frame=glob->prev_frame;
    if (glob->sys_time>DURATION) (
        glob->run_time=0;
        mode!=flag_wave!flag_show;
    ) else
         if (glob->run_time==0) (
             glob->run_time=glob->sync_time-glob->sys_time;
             glob->run_frame=glob->prev_frame;
             mode!=flag_wave!flag_show;
         } else {
             unsigned long frame, late;
             frame=glob->run_frame+(glob->sync_time-glob->run_time)/DURATIO
             late=(glob->sync_time-glob->run_time) %DURATION:
             if (frame<=glob->prev_frame)
                 mode:=flag_wave:flag_show:
             if (frame<=glob->prev_frame && late+glob->tree_time+glob->wave
                 mode:=flag_wave;flag_show;*/
    break;
switch(frmh->flags) (
case KFH_SKIP:
    DebugHsg(*\pModeSwitch - jump/skip*);
    glob->run_time=0;
    break:
case KFH_INTRA:
    DebugHsg(*\pModeSwitch - jump/intra*);
    mode=flag_tree!flag_wave!flag_show!flag_full:
    for(i=glob->prev_frame:i<frmh->frame_number:1++)
```

FILE

·fp;

```
Engineering:KlicsCode:CompPict:KlicsCodec.c
                glob->out(frmh->frame_number%15)=0;
            glob->prev_frame=frmh->frame_number:
            glob->real_frame=glob->prev_frame;
            glob->run_time=0;
            break;
        default:
            DebugMsg(*\pModeSwitch - jump/inter*);
            glob->run_time=0;
            break;
    DebugMsg('\pModeSwitch - display info');
*ifndef COMPONENT
   glob->out(frmh->frame_number%15] = (mode&flag_show) !=0;
    for(i=0,fps=0;i<15;i++) if (glob->out(i]) fps++;
    GetPort((GrafPtr *)&savePort);
    GetCWMgrPort(&wPort);
    SetPort((GrafPtr)wPort);
    SetRect(&rect, 0, 20, 120, 50);
    ClipRect(&rect);
    EraseRect(&rect);
    NumToString(frmh->frame_number,str);
    MoveTo(0,35);
    DrawString(str);
    DrawString("\p:");
    NumToString(fps, str);
    DrawString(str);
    MoveTo(0,50);
    for(i=0;i<15;i++)
        if (glob->out(i)) DrawString('\pX');
        else DrawString(*\pO*);
    SetRect(&rect, -2000, 0, 2000, 2000);
ClipRect(&rect);
    SetPort((GrafPtr)savePort);*/
#endif
    DebugMag(*\pModeSwitch - end*);
    return(mode);
)
#ifndef ENCODER
pascal long
KLBandDecompress(Handle storage, register CodecDecompressParams *p)
*pragma unused(storage)
   Globals **glob = (Globals **)storage;
    ImageDescription
                         *'desc = p->imageDescription;
    int
                         x.y:
    char
                         *baseAddr;
    short
                         rowBytes;
    Rect
                        dRect;
    long
                        offsetH, offsetV;
   OSErr
                         result = noErr;
                         *src(3), *dst(3);
    short
    long
                         *pixmap;
                        width=(*desc)->width+KLExtendWidth((*desc)->width);
    int
                        height=("desc)->height+KLExtendHeight(("desc)->height);
    int
    int
                        hwidth=width>>1.hheight=height>>1,area=neight*width;
    KlicsE
                        kle:
                        *fruh;
   KlicsPrameHeader
    char
                        mmuMode=1;
    long
                        mode:
    SharedGlobals
                         *sGlob;
```

```
Engineering:KlicsCode:CompPict:KlicsCodec.c
    char
                          file_name(30);
    CGrafPtr ==
                         wPort. savePort:
    Rect
                          rect:
    Str255
                          SET:
    HLock((Handle)glob);
    DebugMsg(*\pBandDecompress*);
    (*glob)->sys_time=GetTimerValue(&(*glob)->sys_time);
    (*glob)->sys_time-=(*glob)->sync_time;
*ifdet PERFORMANCE
    (void) PerfControl(ThePGlobals, true);
*endif
    kle=&(*glob)->kle;
    sGlob=(*glob)->sharedGlob;
    dRect = p->srcRect;
    if ( !TransformRect(p->matrix.&dRect.nil) ) (
        DebugMsg(*\pTransformRect Error*);
        return(paramErr);
    rowBytes = p->dstPixHap.rowBytes & 0x3fff:
    offsetH = (dRect.left - p->dstPixMap.bounds.left);
    switch ( p->dstPixMap.pixelSize ) {
    case 32:
        offsetH <<=2:
        break:
    case 16:
        offsetH <<=1;
        break:
    case 8:
        break;
    default:
        result = codecErr:
        DebugHsg(*\pDepth Error*);
       goto bail:
   offsetV = (dRect.top - p->dstFixMap.bounds.top) * rowEytes:
   baseAddr = p->dstPixMap.baseAddr + offsetM + offsetV;
pixmap=(long *)baseAddr;
    Klics decode
   src[0] = (*glob) ->src[0]; src[1] = (*glob) ->src[1]; src[2] = (*glob) ->src[2];
dst[0] = (*glob) ->dst[0]; dst[1] = (*glob) ->dst[1]; dst[2] = (*glob) ->dst[2];
   frmh=(KlicsPrameHeader *)p->data:
   kle->buf.buf=(unsigned long *)(p->data+sizeof(KlicsFrameHeader));
   mode=ModeSwitch(*glob, frmh);
   KlicsDecode(src,dst,&kle->segh,frmh.&kle->buf,mode,(*glob)->scale,&(*glob)->tr
   if ( kle->buf.ptr-kle->buf.buf > frmh->length+2)
       DebugMsg('\pWarning: Decompressor read passed end of buffer');
   p->data(0)='X';
   p->data(1)=mode&flag_tree?'T':'.';
```

```
Engineering:KlicsCode:CompPict:KlicsCodec.c
    p->data(2)=mode&flag_wave?'W':'';
   p->data(3)=mode&flag_show?'S':
   p->data(4)=mode&flag_full?'F':'
   p->data(5)=frmh->flags&KFH_INTRA?'I':'';
   p->data(6)=frmh->flags&KFH_SKIP?'K': ':
    p->data(7)='X';
   p->data-=p->bufferSize;
       signed 10 bit YUV-unsigned 8 RGB convert
*ifdef COMPONENT
   SwapMMUMode (&mmuMode);
*endif
   if (mode&flag_show) (
        (*glob) ->sync_time=GetTimerValue(&(*glob) ->sync_time);
        (*glob) ->dpy_frame=(*glob) ->real_frame;
       if ((*glob)->scale(2)<(*glob)->scale(1)) {
           switch(kle->seqh.channels) {
            case 3:
               switch (p->dstPixMap.pixelSize) (
               case 32:
                    if (mode&flag_full)
                       OUT32X2(sGlob->tab(Use32-1),pixmap,src(0),src(1),src(2),wi-
                       OUT32X2D(sGlob->tab(Use32-1).pixmap.src(0).src(1).src(2).w
                   break;
               case 16:
                   OUT16X2(sGlob->cab(Use16-1),pixmap,src[0],src[1],src[2],width>
                   break;
               case 8:
                   OUT8X2(sGlob->tab(Use8-1),pixmap,src[0],src[1],src[2],width>>(
                   break:
               break;
       ) else (
           switch(kle->seqh.channels) (
           case 3:
               switch (p->dstPixHap.pixelSize) {
               case 32:
                   if (mode&flag_full)
                       OUT32(sclob->tab(Use32-1),pi_map,src[0],src[1],src[2],widt
                   e)8e
                       OUT32D(sGlob->tab(Use32-1),pixmap,src(0),src(1),src(2),wid
                   break;
               case 16:
                   OUT16(sGlob->tab(Use16-1),pixmap,src(0),src(1),src(2),width>>(
                   break;
                   OUT8(sGlob->tab(Use8-1),pixmap,src[0],src[1],src[2],width>>(*g
                   break;
               break;
           }
       (*glob)->dpy_time=GetTimerValue(&(*glob)->dpy_time);
       (*glob)->dpy_time-=(*glob)->sync_time;
  )
```

Engineering: KlicsCode: CompPict: Klics.h

```
© Copyright 1993 KLICS Limited
 * All rights reserved.
    Written by: Adrian Lewis

    Second generation header file

#include
             <stdio.h>
/* useful X definitions */
/*typedef char Book
typedef char *String;
                    Boolean: */
#define True
*define False
/* new Blk definition */
typedef int
                Blk[4];
*define WT_Haar 0
#define WT_Daub4 1
/* mode constructors */
#define M_LPF 1 #define M_STILL 2
*define M_SEND 4 *define M_STOP 8
#define M_VOID 16
#define M_QUIT 32
/* LookAhead histogram */
#define HISTO 300
#define HISTO_DELTA 15.0
#define HISTO_BITS 10
/* Fast Functions */
/* Is the block all zero ? */
#define BlkZero(block) \
    block[0]==0 && block[1]==0 && block[2]==0 && block[3]==0
/* Sum of the absolute values */
*define Decide(new) \
    abs(new(0))+ \
    abs(new[1])+ \
    abs(new[2])+ \
    abs(new(3))
/* Sum of the absolute differences */
#define DecideDelta(new,old) \
    abs(new[0]-old(0])+ \
    abs(new[1]-old[1])+ \
abs(new[2]-old[2])+ \
    abs(new(3)-old(3))
/* Adjust the norm for comparison with SigmaAbs */
#define DecideDouble(norm) (4.0*norm)
/* Get addresses from x,y coords of block. sub-band, octave.
```

```
Engineering:KlicsCode:CompPict:Klics.h

    image size and mask (directly related to octave) information

 *define GetAddr(addr.x,y,sub.oct.size.mask) \(\)
 inc smask=mask>>1, \
         x0=xi(sub41?smask:0), y=
        xl=x1(sub&1?smask:0) imask. \
         y0=(y1(sub&2?smask:0))*size(0),
         yl=(y|(sub&2?smask:0)|mask)*size(0); \
     addr(0)=x0-y0; \
     addr{1}=x1+y0; \
     addr[2]=x0+y1; \
     addr(3)=x1+y1; \
/* Get data values from addresses and memory */
#define GetData(addr.block,data) \
    block[0]=(int)data[addr[0]]; \
    block(1)=(int)data(addr(1)); \
    block[2]=(int)data[addr[2]); \
    block[3] = (int)data[addr[3]];
#define VerifyData(block.mask.tmp) \
    tmp=block&mask: \
     if (tmp!=0 && tmp!=mask) ( \
         block=block<0?mask:-mask: \
/* Pur data values to memory using addresses */
*define PutData(addr.block.data) \
    data(addr(0))=(short)block(0);
    data(addr(1))=(short)block(1); \
    data(addr[2])=(short)block(2); \
    data(addr(3))=(short)block(3);
/* Put zero's to memory using addresses */
#define PutZero(addr.data) \
    data[addr[0]]=0; \
    data[addr[1]]=0; \
data[addr[2]]=0; \
    data(addr(3))=0;
/* Mode: H_VOID Put zero's and find new mode */
#define DoZero(addr.dst.mode.oct) \
    PutZero(addr.dst): \
    mode(oct)=oct==0?M_STOP:M_VOID:
/* Descend the tree structure
 * Copy mode, decrement octave (& mask), set branch to zero
#define DownCounters(mode.oct.mask,blk) \
    mode(oct-1) = mode(oct); \
    oct--; \
    mask = mask>>1; \
   blk(oct)=0;
/* Ascend the tree structure
 * Ascend tree (if possible) until branch not 3

    If at top then set mode to M_QUIT
    Else increment branch and x, y coords

#define StopCounters(mode.oct,mask.blk.x,y.octs) \
    while(oct<octs-1 && blk(oct)==3) ( \
```

Engineering:KlicsCode:CompPict:Klics.h

```
blk{oct}=Q: \
   mask= mask<<1; \
   x &= -mask; \
   y &= -mask; \
   oct++; \

if (oct==octs-1) mode{oct}=M_QUIT; \
  else { \
        blk{oct}++; \
        x ^= mask<<1; \
        if (blk{oct}==2) y ^= mask<<1; \
        mode{oct}=mode{oct+1}; \
}</pre>
```

Engineering: KlicsCode: CompPict: Haar.a

```
© Copyright 1993 KLICS Limited
   All rights reserved.
   Written by: Adrian Lewis
    68000 FastForward/Backward Haar
        macro
        Fwd0
                    &addr0,&dG,&dH
        move.w
                    (&addr0),&dG
                                    ; dG=*(short *)addrl
                                     ; dH=dG
        move.w
                    &dG, &dH
        endm
        macro
        Fwd1
                    &addrl,&addr0,&dG,&dH
                                     ; v=*(short *)acdr2
        move.w
                     (&addrl),d0
                                     : dH+=v
        add.w
                    d0.LdH
                                     : dG-=v
        sub.w
                    d0.&dC
        clr.w
                    d0
                                     ; d0=0
        asr.w
                    ♦1,6dH
                                     ; dAH>>=1
        addx.w
                    do, LdH
                                     ; round dH
                    *1.4dG
                                     ; dG>>=1
        asr.w
                                    ; round dG
; *(ahort *)addr0=dH
        addx.w
                    d0,&dG
                    &dH. (&addr0)
        move.w
                                     : *(short *)addrl=dG
                    &dG, (&addr1)
        move.w
        mend
                    &base, &end, &inc
                                             ; addr0=base
        movea.l
                    &base, a0
                                             ; d0=inc
        move.1
                    &inc.d0
                    #1,d0
a0.a1
        asr.l
                                             ; d0=inc>>1
                                             ; addrl=addr0
        movea.1
                                             ; addrl-=(inc>>l)
                    d0.al
        suba. 1
edo
        Fwd0
                                             ; Fwd0 (addr0.dG.dH)
                    a0,d4,d5
        adda.l
                                             ; addrl+=inc
                    &inc,al
                                             ; Fwdl(addrl,addr0.dG,dH)
        Fwdl
                    al,a0,d4,d5
        adda.1
                    &inc.a0
                                             ; addr0+=inc
        cmpa.l
                    a0,&end
                                             ; addr0<end
        bgt.s
                    @do
                                             ; while
HaarPorward FUNC
                  EXPORT
                                             ; no local variables
        link
                    a6,#0
                    d4-d7/a3-a5,-(a7)
                                             ; store registers
        movem.1
        move.1
                    $000C(a6),d3
                                             ; inc=incl
        movea.1
                    $0008(a6),a5
                                             : base=data
        move.1
                    $0010(a6),d6
                                             ; endl
        move.1
                    $0018(a6),d7
                                             ; end2
                    50014(a6),d2
        move.1
                                             ; inc2
```

Engineering: KlicsCode: CompPict: Haar.a

```
₹ a5,a4
                                                : end=base
        movea.l
@do
                                                ; end+=endl
                      d6.a4
        adda.l
        Fud
                      a5,a4,d3
                                                : Fwd(base.end.inc)
                                                : base+=inc2
                      d2,a5
        adda.l
                                                : end2>base
        cmpa.l
                      d7.a5
        blt.s
                      @do
                                                : for
                                                ; restore registers
                      (a7)+,d4-d7/a3-a5
        movem.l
                                                ; remove locals
        unlk
        rts
                                                : return
        ENDFUNC
        macro
                     '&addr0,&dG,&dH
         Bwd0
                      (&addr0),&dG ; dG=*(short *)addr0 &dG,&dH ; dH=dG
         move.w
         move.w
         endm
         macro
                      &addrl,&addr0,&dG,&dH
         Bwdl
                                       ; v=*(short *)addrl
                       (&addr1),d0
         move.w
                                       ; dH+=v
         add.w
                      ਰ0, ਵਰਮ
                                       ; dG-ev
                      do, ado
         sub.w
                                      ; *(short *)addr0=dH
; *(short *)addr1=dG
                      &dH. (&addr0;
         move.w
                     EdG, (&addr1)
         move.w
         endm
         macro
         Bwd
                      &base, &count, &inc
                                                ; addr0=base
         movea.1
                      &base, a0
                                                ; d0=inc ·
                      &inc.d0
         move.1
                                                 ; d0=inc>>1
                      #1,d0
         asr.l
                                                 ; addri=addr0
                      a0,a1
d0,a1
         movea.1
                                                 ; addr1-=(inc>>1)"
         suba.l
                                                ; Bwd0 (addr0,dG,dH)
                      a0,d4,d5
 @do
         Bwd0
                                                ; addrl+=inc
                      &inc.al
         adda.1
                                                ; Bwd1(addr1,addr0,dG,dH)
                      a1, a0, d4, d5
         Bwd1
                                                ; addr0+=inc
         adda.1
                      &inc.a0
                      &count, 9do
                                                ; while -1!=count
          dbf
          endm
 HaarBackward
                FUNC EXPORT
     d0 - spare, d1 - count1, d2 - inc2, d3 - inc1, d4 - dG, d5 - dH, d6 - loop1, d
                                                 ; no local variables
                       a6,#0
          link
                                                 ; store registers
                       d4-d7/a3-a5,-(a7)
          movem.l
                                                 ; inc=incl
                       $000C(a6),d3
          move.1
                       $0008 (a6) , a5
                                                 ; base=data
          movea.1
                                                 ; loop1 (width/height)
                       $0010 (a6) .d6
          move.1
                                                ; loop2 (height/width); inc2
                       $0018 (a6),d7
          move.1
                       $0014 (a6),d2
          move.1
                                                ; loop2-=1
                       #1,d7
          subq.1
                                                ; loop1/=2
; loop1-=1
                       #1.d6
          lsr.l
          subq.1
                       #1.d6
```

Engineering: KlicsCode: CompPict: Haar. a

```
~d6.d1
                                               : countl=loopi
3do
        move.l
                                               : Bwd(base.count.inc)
                     a5.d1.d3
        Bwd
                                               ; base+=inc2
                     d2.a5
        adda.l
                     d7, @do
                                               ; while -1!=--loop2
        db£
                                               : restore registers
                     (a7)+.d4-d7/a3-a5
        movem.1
                                               : remove locals
                     a6
        unlk
        ENDFUNC
                     EXPORT
HaarXTopBwd FUNC
                                               ; no local variables
         link
                     a6,#0
                     $0008(a6),a0
                                               ; start
         movea.1
                                               ; area
                     $000C(a6),d3
         move.1
                                               ; area (long)
         lsr.l
                      #1.d3
                      #1,d3
                                               ; area-=1
         subq.1
                                               ; d0=HG=*Y
                      (a0), d0
@do
         move.1
                                               ; dl=HG
                     d0,d1 -
         move.1
                                               ; dl=GH
         swap
                     d1
                                               ; d0=H(-G)
                      d0
         neg.w
                                               ; d0=01
         add.1
                      d1,d0
                                               '; *Y++=01
                      d0,(a0)+
         move.l
                                               ; while -l!=--area
                      d3,8do
         dbf
                                               ; remove locals
         unlk
                                               : return
         rts
         ENDFUNC
HaarTopBwd FUNC
                      EXPORT
                                               ; no local variables
                      a6,#0
         link
                                               ; store registers
                      d4-d6.-(a7)
         movem.l
                      $0008(a6),a0
                                               ; startH
         movea.1
                      a0,a1
$000C(a6),d< -
                                                 startG
         movea.l
                                                ; height
         move.l
                                                ; width
                      $0010(a6),d3
         move.l
                                                ; linelen=width
         move.1
                    - d3,d6
                                                ; linelen (bytes)
         add.l
                      d6,d6
                                               ; height/=2
         lsr.l
                      #1,d4
                                                 width/=2
                      #1.d3
         isr.1
                                                : height-=1
                      #1,d4
         suba. 1
                                                 width-=1
         subq. 1
                      #1,d3
                                                ; startG+=linelen
                      d6,a1
         adda.l
 @dol
                                                ; linecount=width
                      d3,d5
         move. 1
                                                : d0=HAHB=*YO
                      (a0),d0
 edo2
         move. 1
                                                ; dl=GAGB=*Yl
                      (a1),d1
         move.l
                      d0,d2
                                                ; d2=HAHB
         move.1
                                                ; d0=0A0B
                      d1,d0
         add.1
                                                ; d2=1A1B
                      d1,d2
         sub.1
                                                : d1=HG
                      d0.d1
         move.1
                                                ; dl=GH
         swap
                      dl
                                                : d0=H(-G)
                      d0
         neg.w
                                                ; d0=01
                      d1.d0
         add.l
                                                : *Y0++=0A0B
                      d0. (a0)+
         move. 1
                                                ; dl=HG
                      d2.d1
         move.l
                      d1
                                                : d1=GH
          swap
```

- 711 -

Engineering:KlicsCode:CompPict:Haar.a

neg.∵	= d2		: d2=H(-G)
add.1	d1.d2		: d2=01
move. 1	d2.(a1)+		: *Y1++=1A1B
dbf move.l dbf	d5.9do2 a1.a0 d4.9do1	•	<pre>: while -1!=linecount : startH=startS : while -1!=height</pre>
movem.l unlk rts	(a7)+,d4-d6 a6		<pre>; restore registers ; remove locals ; return</pre>
 ENDFUNC			
 END			

```
Engineering:KlicsCode:CompPict:ConvolveSH3.c
    $ Copyright 1993 KLICS Limited
All rights reserved.
    Written by: Adrian Lewis
 2D wavelet transform convolver (fast hardware emulation)
    New improved wavelet coeffs : 11 19 5 3
    Optimized for speed:
        dirn - False
        src/dst octave == 0
#define FwdS(addr0,dAG,dAH) \
    v=*(short *)addr0: \
    dAG=(v3=v+(vs=v<<1)); \
    dAG+=v+(vs<<=1); \
    dAH=v3+(vs<<=1); \
    dAH+=v3+(vs<<=1);
"define Fwdl(addrl,dAG,dAH,dBG,dBH) \
    v="(shor: ')addrl; \
    dBG=(v3=v+(vs=v<<1)); \
    dAH+=v+(vs<<=1); \
    dBH=v3+(vs<<=1); \
    dAG-=v3+(vs<<=1);
#define Fwd2(addr2.addr1.addr0.dAG.dAH.dBG.dBH) \
    v=*(short *)addr2; \
dAH-=(v3=v+(vs=v<<1)); \</pre>
    dBG+=v+(vs<<=1); \
    dAG+=v3+(vs<<=1); \
    dBH+=v3+(vs<<=1); \
    *(short *;addr0=(dAH+15)>>5; \
    *(short *)addr1=(dAG+15)>>5;
#define Fwd3(addr3,dAG,dAH,dBG,dBH) \
    v=*(shor: *)addr3; \
    dAG=(v3=v+(vs=v<<1)); \
    dBH+=v+(vs<<=i); \
    dAH=v3+(vs<<=1); \
    dBG-=v3+(vs<<=1);
#define Fwd0(addr0.addr3.addr2.dAG.dAH.dBG.dBH) \
    v=*(short *)addr0; \
    dBH-=(v3=v+(vs=v<<1)); \
   dAG+=v+(vs<<=1); \
    dBG+=v3+(vs<<=1); \
    dAH+=v3+(vs<<=1); \
    *(short *)addr2=(dBH+15)>>5; \
    *(short *)addr3=(dBG+15)>>5;
*define FwdE(addr3.addr2.dBG.dBH) \
   v=*(short *)addr3; \
    dBH+=(vs=v<<1); \
   dBG-=(vs<<2); \
   *(short *)addr2=(dBH+15)>>5; \
*(short *)addr3=(dBG+15)>>5;
```

```
Engineering: KlicsCode: CompPict: ConvolveSH3.c
```

```
*define Fwd(base, endrinc) \
    addr0=base: \
     addr3=addr0-(inc>>2); \
     addr2=addr3-(inc>>2); \
     addrl=addr2-(inc>>2); \
     FwdS(addr0,dAG,dAH); \
     addrl+=inc; \
    Fwdl(addr1,dAG,dAH,dBG,dBH); \
     addr2+=inc: \
     Fwd2 (addr2, addr1, addr0, dAG, dAH, dBG, dBH); \
    addr3+=inc; \
    while(addr3<end) ( \
         Fwd3(addr3,dAG,dAH,dBG,dBH); \
         addr0+=inc: \
         Fwd0(addr0,addr3,addr2,dAG,dAH,dBG,dBH); \
         addr1+=inc; \
         Fwd1(addr1,dAG,dAH,dBG,dBH); \
         addr2+=inc; \
         Fwd2(addr2,addr1,addr0,dAG,dAH,dBG,dBH); \
         addr3+=inc: \.
    FwdE(addr3,addr2,dBG,dBH);
extern void FASTFORWARD(char *data, long incl. long endl. long inc2, char *end2); extern void HAARFORWARD(char *data, long incl. long endl. long inc2, char *end2);
void
         FastForward(char *data, long incl, long endi, long inc2, char *end2)
    register short v. vs. v3, dAG, dAH, dBG, dBH, inc. register char "addr0, "addr1, "addr2, "addr3, "end;
    char
             *base;
    inc=incl:
    for(base=daca; base<end2; base+=inc2) (</pre>
         end=base+endl;
         Fwd(base, end.inc);
)
        Daub4Forward(short *data. int size(2), int oct_dst;
void
             oct. area=size(0)*size(1)<<1;
    int
             width=size[0]<<1;
    short
             *top=area+(char *)data, *left=width+(char *)data;
    char
    for(oct=0;oct!=oct_dst;oct++) (
               cinc=2<<oct, cinc4=cinc<<2.
rinc=size(0)<<oct+1, rinc4=rinc<<2; /* col and row increments in t.</pre>
        long
        FASTFORWARD((char *)data,cinc4,width-cinc.rinc.top);
        FASTFORWARD((char *)data,rinc4.area-rinc.cinc.left);
    )
}
        HaarForward(short *data, int size(2), int oct_dst)
void
    int
             oct, area=size(0)*size(1)<<1:
             width=size(0)<<1;
    short
    char
             *top=area+(char *)data, *left=width+(char *)data;
    for(oct=0;oct!=oct_dst;oct++) (
        long
                cinc=2<<oct, cinc2=cinc<<1.
```

```
Engineering:KlicsCode:CompPict:ConvolveSH3.c
                 rinc=size(0)<<oct+1. rinc2=rinc<<1; /* col and row increments in t
        HAARFORWARD((char *)data,cinc2,width,rinc.top);
        HAARFORWARD((cnar *)data.rinc2.area.cinc.left;
    }
        HybridForward(short *data, int size(2), int oct_dst)
void
    int
             oct, area=size(0)*size(1)<<1:
    short
             width=size(0)<<1;
    char
             *top=area+(char *)data, *left=width+(char *)data;
    HAARFORWARD((char *)data, 4, width, size(0)<<1, top);</pre>
    HAARFORWARD((char *|data,size(0)<<2,area,2,left);
    for(oct=1;oct!=oct_dst;oct++) (
        long
               cinc=2<<oct. cinc4=cinc<<2.
                 rinc=size[0]<<oct+1. rinc4=rinc<<2; /* col and row increments in t
        FASTFORWARD((char *)data.cinc4,width-cinc.rinc.top);
FASTFORWARD((char *)data.rinc4.area-rinc.cinc.left);
    )
١
*define BwdS0(addr0.dAG,dAH,dBH) \
    v=*(short *)addr0; \
    dAG= -(v3=v+(vs=v<<1)); \
    dAH=v+(vs<<=1); \
    dBH=vs<<1; \
#define BwdS1(addr1,addr0.dAG,dAH,dBH) \
    v=*(short *)addrl: \
    dBH+=(vs=v<<1); \
    v3=vs+v; \
    dAG+=v3+(vs<<=2); \
    dAH-=v3+(vs<<=1); \
    *(short *)addr0=(dBH+3)>>3;
#define Bwd2(addr2,dAG,dAH,dBG,dBH) \
    v=*(short *)addr2; \
    dBG= -(v3=v+(vs=v<<1)); \
    dBH=v+(vs<<=1); \
    dAH+=v3+(vs<<=1); \
    dAG+=v3+(vs<<=1);
#define Bwd3(addr3.addr2.addr1.dAG,dAH,dBG,dBH) \
    v=*(short *)addr3: \
    dAH+=(v3=v+(vs=v<<1)); \
    dAG+=v+(vs<<=1); \
    dBG+=v3+(vs<<=1);
    dBH-=v3+(vs<<=1); \
    *(short *)addr1=(dAH+7)>>4; \
*(short *)addr2=(dAG+7)>>4;
#define Bwd0(addr0,dAG,dAH,dBG,dBH) \
    v=*(short *)addr0; \
    dAG= -(v3=v+(vs=v<<1)); \
    dAH=v+(vs<<=1); \
    dBH+=v3+(vs<<=1); \
    dBG+=v3+(v9<<=1);
#define Bwd1(addr1,addr0,addr3,dAG,dAH,dBG,dBH) \
   v=*(short *)addrl; \
```

Engineering:KlicsCode:CompPict:ConvolveSH3.c

```
dBH+=(v3=v+(vs=k<<1)); \
    dBG+=v+(v8<<=1): \
    dAG+=v3+ivs<<=1); \
    CAH-=v3+(v9<<=1); ;
     *(short *)addr3=(dBH+7)>>4; \
     *(short *)addr0=(dBG+7)>>4;
#define BwdE2(add=2.dAG.dAH.dBH) \
    v=*(short *)addr2; \
    v3=v+(vs=v<<1); \
    dBH=(vs<<=2); \
    dAH+=v3+vs: \
    dAG+=v3+(vs<<=1);
*define BwdE3(addr3,addr2.addr1.dag.daH.dBH) \
    v=*(short *)addr3; \
    dAH+=(v3=v+(vs=v<<1)); \
    dAG+=v+(vs<<=1); \
    dBH-=v3+(vs<<=1); \
    dBH-=v3+(vs<<=1); \
    *(short *)addr1=(dAH+7)>>4; \
    *(short *)addr2=(dAG+7)>>4; \
    *(short *)addr3=(dBH+3)>>3;
*define Bwd(base, end, inc) \
    addr0=base; \
    addr3=addr0-(inc>>2); \
addr2=addr3-(inc>>2); \
    addr1=addr2-(inc>>2);
    BwdS0(addr0,dAG,dAH,dBH); \
    addrl+=inc; \
    BwdS1(addr1.addr0,dAG,dAH,dBH); \
    addr2+=inc; \
    while(addr2<end) ( \
         Bwd2 (addr2, dAG, dAH, dBG, dBH); \
         addr3+=inc: \
         Bwd3 (addr3, addr2, addr1, dAG, dAH, dBG, dBH); \
         addr0+=inc; \
         Bwd0 (addr0.dAG,dAH.dBG,dBH); \
         addrl+=inc: \
         Bwdl (addrl, addrl, addrl, dAG, dAH, dBG, dBH); \
        addr2+=inc: \
    BwdE2(addr2,dAG,dAH,dBH); \
    addr3+=inc: \
    BwdE3(addr3,addr2,addr1,dAG,dAH,dPH);
extern void FASTBACKWARD(char *data, long incl. long loop1, long inc2, char *end2) extern void HAARBACKWARD(char *data, long incl. long loop1, long inc2, long loop2)
extern void HAARTOPEWD(char *data,long height.long width);
/* extern void HAARXTOPBWD(char *data,long area); */
        FastBackward(char *data, long incl. long endl, long inc2, char *end2)
void
    register short v, vs. v3, dAG, dAH, dBG, dBH, inc: register char *addr0, *addr1, *addr2, *addr3, *end;
             "base:
    char
    inc=incl:
    for(base=data; base<end2; base+=inc2) (</pre>
        end=base+endl:
        Bwd(base, end, inc);
```

- 716 -

```
Engineering:KlicsCode:CompPict:ConvolveSH3.c
)
        Daub4Backward(short *data.int size(?).int oct_src)
volá
            oct, area=size(0)*size(1)<<1;
    เกเ
            width=size(0)<<1;
    short
             *top=area+(char *)data. *left=width+(char *)data:
    char
    for(oct=oct_src-1:oct>=0:oct--) (
                 cinc=2<<oct, cinc4=cinc<<2.
        long
                 rinc=size[0]<<oct+1, rinc4=rinc<<2: /* col and row increments in t
        FASTBACKWARD((char *)data.rinc4.area-(rinc<<1).cinc.left);</pre>
        FASTBACKWARD((char *)data.cinc4.width-(cinc<<1).rinc.cop);</pre>
١
void
        HaarBackward(data, size, oct_src)
short
        *data:
int
        size(2), oct_src;
    int
             oct, area=size(0)*size(1)<<1;
            width=size(0)<<1;
    Short
             *top=area+(char *)data, *left=width+(char *)data;
    char
    for(cct=oct_src-1;oct>0;oct--) (
                 cinz=2<<oct, cinc2=cinc<<1.
rinc=stre[0]<<oct+1, rinc2=rinc<<1; /* col and row increments in t</pre>
        long
        HAARBACKWARD((char *)data.rinc2,size(1)>>oct.cinc.size(0)>>oct);
        HAARBACKWARD((char *)data.cinc2, size[0]>>oct.rinc.size[1]>>oct);
    HAARTOPBWD((char *)data.size(1),size(0));
    HAARXTOPBWD((char *)data.area>>1);*/
void
        HybridBackward (data, size, oct_src)
short
        'data:
        size(2), oct_src;
int
             oct. area=size(0)*size(1)<<1;</pre>
    int
            width=size(0)<<1;
    short
             *top=area+(char *)data, *left=width+(char *)data;
    char
    for(oct=oct_src-1;oct>0;oct--) (
                 cinc=2<<oct, cinc4=cinc<<2,
                 rinc=size(0)<<oct+1, rinc4=rinc<<2; /* col and row increments in t
        FASTBACKWARD((char *)data.rinc4.area-(rinc<<1).cinc.left);
FASTBACKWARD((char *)data.cinc4.width-(cinc<<1).rinc.top);</pre>
    HAARTOPBWD((char *)data, size[1], size[0]);
    HAARXTOPBWD((char *)data,area>>1); */
```

Engineering: KlicsCode: CompPict: ConvolveSH3.a

```
© Copyright 1993 KLICS Limited
All rights reserved.
Written by: Adrian Lewis
68000 FastForward/Backward code
    seg
                'klics'
    macro
    FwdStart
                &addrC.&dAG,&dAH
    move.w
                 (&addr0),d0
                                 ; v=*(short *)addr0
    move.w
                 d0,d1
                                  ; vs=v
    add.w
                 dl,dl
                                  ; vs<<=1
    move.w
                 d1.d2
                                  ; v3=vs
    add.w
                 d0,d2
                                  ; v3=vs+v
    move.w
                 d2, &dAG
                                  ; dAG=v3
    add.w
                 d1,d1
                                  ; vs<<=1
    add.w
                 do, &dAG
                                  ; dAG+=V
                 dl.&dAG
    add.w
                                  ; dAG+=vs
                 d2,&dAH
    move.w
                                  ; dAH=v3
    add.w
                 dl,dl
                                  ; vs<<=1
    add.w
                 HADA, ID
                                  ; dAH+=vs
    add.w
                 d2, EdAH
                                  ; dAH+=v3
    add.w
                 d1.d1
                                 ; vs<<=1
    add.w
                 d1, &dAH
    ಹುರೂ
    macro
    PwdOdd
                 &addr1, &dAG, &dAH, &dBG, &dBH
                 (&addr1),d0
    move.w
                                 ; v=*(short *)addr1
    move.w
                d0.d1
                                 ; VS=V
    add.w
                d1.d1
d1.d2
                                 ; VS<<=1
    move.w
                                 ; v3=vs
    add.w
                 d0.d2
                                 ; v3=vs+v
    move.w
                d2, &dBG
                                 ; dBG=v3
    add.w
                d1.d1
                                 ; vs<<=1
    add.w
                do, Edah
                                 ; dAH+=v
    add.w
                dl.&dAH
                                 ; dAH+=vs
                d2,&dBH
    move.w
                                 ; dBH=v3
    add.w
                dl,dl
                                 ; V5<<=1
                dl.&dBH
    add.w
                                 ; dBH+=vs
    sub.w
                d2,&dAG
                                 ; dAG-=v3
    add.w
                 d1,d1
                                 ; V5<<=1
                dl, £dAG
    sub.w
                                 ; dAG-=vs
    endm
    macro
    PwdEven
                &addr2.&addr1,&addr0.&dAG,&dAH,&dBG,&dBH
    move.w
                 (&addr2),d0
                                 ; v=*(short *)addr2
                                 ; VS=V
; VS<<=1
    move.w
                d0,d1
                d1,d1
d1,d2
    add.w
    move.w
                                 ; v3=vs
```

Engineering: KlicsCode: CompPict: ConvolveSH3.a

```
₹ d0.d2
        add.w
                                        : v3=vs+v
        sub.w
                      d2.&dAH
                                        ذ dAH-=v ذ
        add.w
                      dl.dl
                                        ; vs<<=1
        add.w
                      do, EdBC
                                          dBG+=v
        add.w
                      d1.&dBG
                                        : dBG+=vs
        add.w
                      d2.&dAG
                                          dAG+=v3
        add.w
                      dl.dl
                                          vs<<=1
        add.w
                      dl. EdAG
                                        ; dAG+=vs
                      d2,&dBH
        add.w
                                          dBH+=v3
        add.w
                      dl.dl
                                          vs<<=1
        add.w
                      d1.&dBH
                                       ; dBH+=vs
        clr.w
                      d0
                                        ; d0=0
                      #5.&dAH
        asr.w
                                       ; dAH>>=5
        addx.w
                      HADS, 0D
                                        ; round dAH
        asr.w
                      $5,&dAG
                                       ; dAG>>=5
                                       ; round dAG
        addx.w
                      dO. &dAG
        move.w
                                       ; *(short *)addr0=dAH
; *(short *)addr1=dAG
                      &dAH. (&addr0)
        move.w
                      &dAG, (&addrl)
        mend
        macro
        FwdEnd
                &addr3,&addr2,&dBG,&dBH
        move.w
                      (&addr3),d0
                                        ; v=*(short *)addr3
        add.w
                      d0,d0
                                       ; v<<=1
        add.w
                      d0.&dBH
                                       ; dBH+=v
         lsl.w
                      #2,d0
                                       : v<<=2
                      d0.&dBG
                                       ; dBG-=v
        sub.w
        clr.w
                                       : d0=0
                      d0
        asr.w
                      #5.&dBH
                                       ; dBH>>=5
        addx.w
                                       : round dBH
                      HED3,0b
                                       ; dBG>>=5
        asr.w
                      #5,&dBG
        addx.w
                      d0,&dBG
                                       ; round dBG
        move.w
                      &dBH. (&addr2)
                                         *(short *)addr2=dBH
        move.w
                      &dBG, (&addr3)
                                       ; *(short *)addr3=dBG
        endm.
        macro
        Fwd
                      &base, &end, &inc
        movea.l
                                                : addr0=base
                      &base, a0
        move.l
                      Linc.d0
                                                : d0=inc
        asr.l
                      #2,d0
                                                 : d0=inc>>2
        movea.1
                      a0, a3
                                                : addr3=addr0
        suba.l
                      d0.a3
                                                ; addr3-=(inc>>2)
        movea. 1
                      a3.a2
                                                ; addr2=addr3
        suba.l
                      d0, a2
                                                : addr2-=(inc>>2)
        movea.1
                      a2,a1
                                                : addrl=addr2
        suba.1
                      d0.al
                                                  addr1-=(inc>>2)
        FwdStart
                      a0,d4,d5
                                                ; FwdStart (addr0,dAG,dAH)
        adda.1
                      &inc.al
                                                  addr1+=inc
        Fwd0dd
                      al, d4, d5, d6, d7
                                                ; FwdOdd (addr1, dAG, dAH, dBG, dBH)
        adda.l
                      &inc,a2
                                                  addr2+=inc
        FwdEven
                                                ; FwdEven(addr2,addr1.addr0,dAG,dAH,dB
                      a2.a1.a0,d4,d5,d6,d7
        adda.1
FwdOdd
                     &inc.a3
a3.d6.d7.d4.d5
                                                  addr3+=inc
0do
                                                  FwdOdd (addr3, dBG, dBH, dAG, dAH)
        adda, 1
                                                  addr0+=inc
                      £inc, 20
        FwdEven
                                                  FwdEven(addr0,addr3,addr2,dBG,dBH,dA-
                      a0,a3,a2,d6,d7,d4,d5
        adda.1
                     &inc,al
                                                  addrl+=inc
        Fwd0dd
                                                  Fwdodd(addr1,dAG,dAH,dBG,dBH)
                     al,d4,d5,d6,d7
        adda.l
                     &inc.a2
                                                  addr2+=inc
```

endm

Engineering: KlicsCode: CompPict: ConvolveSH3.a

```
FwdEven
                      a2.a1.a0,d4,d5,d6.d7
                                               ; FwdEven; addr2, addr1, addr0, dAG, dAH, dB
        adda.1
                      &inc.a3
                                               ; addr3.minc
         cmpa.l
                      a3.Eend
                                               : addr3<end
        bgt.v
                      9do
                                               ; while
        FwdEnd
                     a3.a2.d6.d7
                                               : FwdEnd(addr3.addr2,dBG,dBH)
FastForward FUNC
                     EXPORT
        link
                     a6,#0
                                               ; no local variables
                     d4-d7/a3-a5,-(a7)
        movem.1
                                              ; store registers
        move.1
                     $000C(a6),33
                                               ; inc=incl
        movea.1
                     $0008(a6),a5
                                               ; base=data
900
        movea.1
                     a5.a4
                                               : end=base
        adda.l
                     $0010(a6),a4
                                               : end+=endl
        FWd
                     a5, a4.d3
                                               ; Fwd(base, end, inc)
        adda.l
                     $0014(a6),a5
                                               ; base+=inc2
                     $0018(a6),a5
        cmpa.1
                                               ; end2>base
        blt.w .
                     ಯಾ
                                               ; for
        movem. 1
                     (a7)+,d4-d7/a3-a5
                                              ; restore registers
        unlk
                                               ; remove locals
        rts
                                               ; return
        ENDFUNC
        macro
        BwdStart0
                     &addr0, &dAG, &dAH, &dBH
                     (&addr0),d0
                                      ; v=*(short *)addr0
        move.w
                     d0,d1
                                      : VS=V
        add.w
                     d1,d1
                                      ; vs<<=1 (vs=2v)
        add.w
                     d1,d0
                                      ; V+=VS (V=3V)
                                      : dAG=v3
        move.w
                     do, adag
        neg.w
                     AdAG
                                      ; dAG= -dAG
                     HADA, OD
        move.w
                                      ; dAH=v
        add.w
                     d1, &dAH
                                      ; dAH+=vs
        lsl.w
                     #2.dl
                                      ; vs<<=2 (vs=8v)
        move.w
                     dl.&dBH
                                      : dBH=vs
        endm
        macro
        BwdStart1
                    &addr1,&addr0,&dAG,&dAH,&dBH
                     (£addr1),dG
        move.w
                                      ; v=*(short *)addrl
        move.w
                     d0,d1
                                      ; VS=V
        add.w
                    dl,dl
                                      ; vs<<=1
        add.w
                     dl,&dBH
                                      ; dBH+=vs
                                      ; V+=VS (V=3V)
; V8<<=2 (VS=8V)
        add.w
                     d1,d0
        181.1
                     #2,d1
        add.w
                     d1.d0
                                      ; v+=vs (v=11v)
        add.w
                     dD, &dAG
                                      ; dAG+=v
                                      ; v+=vs (v=19v)
        add.w
                     d1,d0
        sub.w
                    dO, EdAH
                                      ; dAH-=v
                    đĐ
                                      ; d0=0
        clr.w
                     #3, &dBH
        asr. v
                                      ; dBH>>=3
                                    : round dBH
: *(short *)addr0=dBH
                    dO, &dBH
        addx.v
                    &dBH, (&addr0)
        move.v
```

ī

Engineering: KlicsCode: CompPict: ConvolveSH3.a

```
πacro
BwdEven &addr2.&dAG.&dAH.&dBG.&dEH
                             : v=*(short *)addr2
            (&addr2).d0
                             ; vs=v
            d0,d1
Tove.w
                             ; vs<<=1 (vs=2v)
            dl.dl
acd.w
                             ; v+=vs (v=3v)
            d1.d0
add.w
                             : dBG=v
            d0.&dBC
move.w
                             ; dBG= -dBG
             & dBG
neg.w
                             ; dBH=v
             d0.&dBH
move.w
             dl. &dBH
                             ; dBH+=vs
add.w
                             ; vs<<=2 (vs=8v)
             #2.dl
isl.w
                             ; v+=vs (v=11v)
add.w
             d1.d0
                             ; dAH+=V
add.w
             d0,&dAH
                             ; v+=vs (v=19v)
             d1.d0
acc.w
             d0.&dAG
                             ; dAG+=v
add.w
       macro
             &addr3,&addr2,&addr1,&dAG,&dAH,&dBG,&dBH
BwdOdd
             (&addr3).d0
                             ; v=*(short *)addr3
move.w
                             ; vs=v
move.w
             d0.d1
                             ; vs<<=1 (vs=2v)
             dl,dl
add.w
                              ; v+=vs (v=3v)
             d1.d0
 add.w
                            ; dAH+=v
             dO. &dAH
 add.w
             dO, &dAG
                             ; dAG+=V
 add.w
             d1.&dAG
                             : dAG+=VS
 add.w
                             ; vs<<=2 (vs=8v)
             #2,d1
 lsl.w
                             ; v+=vs (v=11v)
             a1.a0
 add.w
                             ; dBG+=V
 add.w
             d0,&dBG
                             ; v+=vs (v=19v)
 add.w
             d1,d0
                             ; dBH-=V
             d0,&dBH
 sub.w
             đO
                              ; d0=0
 clr.w
                             ; dAH>>#4
             #4.EdAH
 asr.w
                             ; round dAH
             HAD&, Ob
 addx.w
                             ; *(short *)addrl=dAH
             &dAH, (&addrl)
 move.w
                             ; dAG>>=4
             #4,&dAG
 asr.w
                             ; round dAG
; *(short *)addr2=dAG
             do, EdAG
 addx.w
             &dAG.(&addr2)
 move.w
 endm
           -----
 macro
             &addr2,&dAG,&dAH,&dBH
 3wdEnd2
                              ; v=*(short *)addr2
              (&addr2).d0
 move.w
              d0.d1
                              ; VS=V
 move.w
              d1,d1
d1,d0
                              ; vs<<=1 (vs=2v)
 add.w
                              ; v+=vs (v=3v)
; vs<<=2 (vs=8v)
 add.w
              #2.dl
 lsl.w
              dl,&dBH
                              ; dBH=vs
 move.w
                              ; v+=v9 (v=11v)
              d1,d0
 add.w
                              ; dAH+=V
              dO.&dAH
 add.w
                              ; v+=vs (v=19v)
 add.w
              d1.d0
                              ; dAG+=V
  add.w
              do, &dAG
  endm
  macro
              faddri, faddri, faddri, fdAG, fdAH, fdBH
  3wdEnd?
```

Engineering:KlicsCode:CompPict:ConvolveSH3.a

```
move.w
                       (&addr3),d0
                                        : V=*(short *)addr3
          move.w
                       d0.d1
                                        ; vsev
          w.bbs
                       d1.d1
                                        : vs<<=1 (vs=2v)
          add.w
                       d1.d0
                                        7 V+=VS (V=3V)
          add.w
                       HADS, 0b
                                        : dAH+=v
          add.w
                      do, &dAG
                                        ; dAG+=v
          add.w
                      d1, &dAG
                                        ; dAG+=vs
          add.w
                      dl.&dBH
                                        : dBH+=vs
          1s1.1
                       #4.dl
                                        ; VS<<=4 (v=32v)
          sub.w
                      d1,&dBH
                                        : dBH-=vs
         clr.w
                      d0
                                        ; d0=0
         asr.w
                      #4.&dAH
                                        : dAH>>=4
         addx.w
                      HADS, 0b
                                        ; round dAH
         move.w
                      &dAH, (&addrl)
                                        ; *(short *)addrl=dAH
         asr.w
                      #4, &dAG
                                        dAG>>=4
         addx.w
                      do. &dag
                                       : round dAG
         move. w
                      &dAG, (&addr2)
                                        : *(short *)addr2=dAG
         asr.w
                      #3,&dBH
                                         dBH>>=3
         addx.w
                      dO, EdBH
                                       ; Tound dBH
         move.w
                      &dBH, (&addr3)
                                       ; *(short *)addr3=dBH
         endm
         macro
         Bwd
                      &base, &end. &inc
        · movea.l
                      Lbase, a0
                                               ; addr0=base
         move.l
                      &inc,d0
                                               : d0=inc
         asr.l
                      42, d0
                                               : d0=inc>>2
         movea.1
                      a0, a3
                                               ; addr3=addr0
                      d0.a3
         suba.1
                                               ; addr3-=(inc>>2)
         movea.1
                      a3,a2
                                               ; addr2=addr3
         suba.l
                     d0.a2
                                               ; addr2-=(inc>>2)
        movea.1
                      a2,a1
                                               ; addr1=addr2
         suba.1
                     d0.al
                                               // addr1-=(inc>>2)
         BwdStart0
                     a0,d4,d5,d7
                                               : BwdStare0 (addr0, dAG, dAH, dBH)
         adda.1
                     &inc,al
                                               : addrl+=inc
         BwdStart1
                     al, a0, d4, d5, d7
                                               ; BwdStart1(addr1.addr0.dAG.dAH.dBH)
         adda.l
                     &inc,a2
                                                 addr2+=inc
0do
                     a2,d4,d5,d6.d7
        BwdEven
                                               : BwdEven (addr2.dAG.dAH.dBG.dBH)
         adda.l
                     &inc,a3
                                                 addr3+=inc
        BwdOdd
                     a3, a2, a1, d4, d5, d6, d7
                                               ; BwdOdd(addr3,addr2,addr1,dAG,dAH,dBG
        adda.1
                     &inc.a0
                                                 addr0+*inc
        BwdEven
                     a0.d6.d7,d4,d5
                                               ; BwdEven(addr0,dBG,dBH,dAG,dAH)
        adda.1
                     &inc,al
                                                addrl+=inc
        BwdOdd
                     al, a0, a3, d6, d7, d4, d5
                                               ; BwdOdd(addr1,addr0,addr3,dBG,dBH,dAG
        adda.l
                     &inc.a2
                                                addr2+=inc
        cmpa.1
                     a2, £end
                                               ; addr2<end
        bgt
                     0do
                                                while
        BwdEnd2
                     a2,d4,d5,d7
                                              ; BwdEnd2 (addr2.dAG,dAH,dBH)
        adda.l
                     &inc.a3
                                              ; addr3+=inc
        BwdEnd1
                     a3, a2, a1, d4, d5, d7
                                              ; BwdEnd3 (addr3, addr2, addr1, dAG, dAH, dB
        endm
FastBackward
              FUNC
                         EXPORT
        link
                     a6, #0
                                              ; no local variables
        movem.1
                     d4-d7/a3-a5,-(a7)
                                              ; Store registers
        move. 1
                     $000C(a6),d3
                                              ; inc=incl
        movea.l
                    $0008(a6),a5
                                              : base=data
```

Engineering:KlicsCode:CompPict:ConvolveSH3.a

		-	
€do	movea.l adda.l Bwd adda.l cmpa.l blc.w	a5,a4 \$0010(a6),a4 a5,a4,d3 \$0014(a6),a5 \$0018(a6),a5 @do	<pre>; end=base : end+=end1 : Bwd(base.end.inc) : base+=inc2 ; end2>base : for</pre>
•	movem.l unlk rcs ENDFUNC	(a7)+.d4-d7/a3-a5 a6	: restore registers : remove locals : return
•	EMT)		

```
Engineering:KlicsCode:CompPict:Colour.c
```

```
Copyright 1993 KLICS Limited
    All rights reserved.
    Written by: Adrian Lewis

    Test versions of colour space conversions in C

finclude <Memory.h>
winclude <QuickDraw.h>
#define NewPointer(ptr.type.size) \
    saveZone=GetZone(); \
SetZone(SystemZone()); \
    if (nil==(ptr=(type)NewPtr(size))) ( \
        Set2one(ApplicZone()); \
         if (nil==(ptr=(type)NewPtr(size))) ( \
             SetZone(saveZone); \
             return(MemoryError()); \
        } \
    SetZone(saveZone);
typedef union (
    long
            pixel:
    char
            rgb(4);
) Pixel:
/* Special YUV space version */
*define rgb_yuv(pixmap,Yc) \
   pixel.pixel=0x808080^*pixmap++; \
    r=(short)pixel.rgb[1]; \
    g=(short)pixel.rgb(2); g+=g; \
    b=(short)pixel.rgb(3); \
   Y=(b<<3)-b; \
    g+=r; \
    Y+=g+g+g; \
    Y>>=4; \
    Y+=g; \
    *YC++=Y; \
    Y>>=2; \
    U+=b-Y; \
    V+=r-Y;
#define limit(Y.low,high) \
    Y<(low<<2)?low<<2:Y>(high<<2)?high<<2:Y
/* Standard YUV space version - Bt294 CR07(0) mode limiting */
*define rgb_yuv32(pixmap,Yc) \
pixel.pixel=0x808080^*pixmap++; \
   r=(long)pixel.rgb[1]; \
    g=(long)pixel.rgb(2); \
    b=(long)pixel.rgb(3); \
   Y= (306°r + 601°g + 117°b)>>8; \
"YC++ = limit(Y,16-128,235-128); \
U+= (512°r - 429°g - 83°b)>>8; \
   V+= (-173°r - 339°g + 512°b)>>8;
        RGB2YUV32/long *pixmap, short *Yc, short *Uc, short *Yc, int area, int wid
```

Engineering:KlicsCode:CompPict:Colour.c

```
*pixmap2*pixmap+cols. *row. *end=pixmap+area;
    long
    short
            'Yc2=Yc+width:
    while(pixmap<end) (
        row=pixmap+width;
        while(pixmap<row) (
            Pixel pixel;
             iong
                     I.g.b,Y,U=0,V=C;
            rgb_yuv32(pixmap, Yc):
            rgb_yuv32(pixmap.Yc);
            rgb_yuv32(pixmap2, Yc2);
            rgb_yuv32(pixmap2, Yc2);
            U>>=2;
            V>>=2:
             *Uc++=limit(U,16-128,240-128);
             *Vc++=limit(V,16-128,240-128);
        pixmap+=cols+cols-width:
        pixmap2+=cols+cols-width;
        Yc+=width:
        Yc2+=width:
    )
typedei struct (
    short ry, rv, by, bu:
RGB_Tap;
OSErr RGBTable(long **tab)
    RGB_Tab *table:
    int
    THZ
            saveZone;
    NewPointer(table,RGB_Tab*,256*sizeof(RGB_Tab));
   *tab=(long *)table;
for(i=0;i<128;i++) (
        table(i).ry=306*i>>8:
        table(i).rv=173*i>>8;
table(i).by=117*i>>8;
        table(i).bu=83*1>>8;
    for(1=128;i<256;i++) (
        table(i).ry=306*(i-256)>>8;
        table(i).rv=173*(i-256)>>8:
        table[i].by=117*(i-256)>>8:
        table(i).bu=83*(i-256)>>8;
   return (noErr);
}
typedef struct {
short ru, gu, bv, gv;
) UV32_Tab;
UV32_Tab *UV32_Table()
    UV32_Tab
                 *table:
    int
            1:
    table=(UV32_Tab *)NewPtr(256*sizeof(UV32_Tab)):
```

```
for(i=C;i<128;i++) {
        table(i).ru=128+(1436*i>>10);
        table(i).gu=128+(-731*i>>10);
        table(i).bv=128+(1815*i>>10):
        table(i).gv=-352*i>>10:
    for(i=128;i<256;i++) {
        table(i).ru=128+(1436*(i-256)>>10);
        table(i).gu=128+(-731*(i-256)>>10);
        table(i).bv=128+(1815*(i-256)>>10);
        table(i).gv=-352*(i-256)>>10;
    return(table);
typedef struct (
    long u, v;
) UV32Tab;
OSErr UV32Table(long **tab)
    long
             'ytab;
    UV32Tab *uvtab;
    int
             i:
    THZ
             saveZone:
    NewPointer(*tab,long*,512*sizeof(long)+512*sizeof(UV32Tab));
    ytab=*tab;
    uvtab=(UV32Tab*)&ytab(512);
    for(i=-256;i<256;i++) (
        long
                yyy, sp;
        sp=0x000000fe4(i<-128?0:i>127?255:i+128);
        yyy=sp; yyy<<=8;
        AAA |=ab: AAA<<=8:</pre>
        yyy I = sp;
        ytab[0x000001ff&i]=yyy;
    for(i=-256;i<256;i++) (
        long
                 ru, gu, bv, gv;
        ru=0xfffffffe & (1436*i>>10);
        gu=0x000001fe & (-731*i>>10);
bv=0x000001fe & (1815*i>>10);
gv=0x000001fe & (-352*i>>10);
        uvtab[0x000001FF&i].u=((ru<<8)|gu)<<8;
        uvtab(0x000001FF&i).v=(gv<<8) |bv:
    return(noErr);
}
typedef struct {
    short u, v;
) UV16Tab;
OSErr UV16Table(long **tab)
    short
             *vtab:
    UV16Tab 'uvtab;
int i;
THz save2one;
```

```
Engineering:KlicsCode:CompPict:Colour.c
   NewPointer(*tab,long*.512*sizeof(short)-512*sizeof(UV16Tab));
   ytab=*(short **)tab;
   uvtab=(UV16Tab*)&ytab(512);
   for(1=-256:i<256:1++) (
               yyy, sp:
       long
       sp=0x0000001e&((i<-129?0:i>127?255:i-1281>>3);
       yyy=sp: yyy<<=5:
       yyy1=sp: yyy<<=5;
       yyy I = sp;
       ytab(0x000001ff&i)=yyy;
   for(i=-256:i<256:i++) (
               ru, gu, bv, gv;
       long
       ru=0xfffffffe & (1436*i>>13);
       gu=0x0000003e & (-731*i>>13);
bv=0x0000003e & (1815*i>>13);
       gv=0x0000003e & (-352*i>>13);
       uvtab(0x000001FF&i).u=((ru<<5)|gu)<<5:
       uvtab(0x000001FF&i).v=(gv<<5) |bv;
   recurn(noErr);
١
*define over(val) \
    ((0xFF00&(val)) == 0)?(char)val:val<0?0:255
/* Standard YUV space version */
#define yuv_rgb32(pixmap,Yc) \
    Y=(*YC++)>>2; \
    pixel.rgb[1] = over(Y+r); \
    pixel.rgb(2)=over(Y+g);
    pixel.rgb(3)=over(Y+b); \
    *pixmap++=pixel.pixel;
        YUV2RGB32(long *pixmap, short *Yc. short *Uc, short *Vc. int area, int wid
void
            *pixmap2*pixmap+cols. *row, *end*pixmap+area;
    long
    short
            *Yc2=Yc+width;
    while(pixmap<end) (
        row=pixmap+width:
        while(pixmap<row) (
            Pixel pixel;
                    r,g,b,Y,U.V;
            long
            U=(*UC++)>>2;
            V=(*Vc++)>>2;
            r=128+(1436*U>>10);
            g=128+(-731*U - 352*V>>10);
             b=128+(1815*V>>10);
            yuv_rgb32(pixmap.Yc);
            yuv_rgb32(pixmap, Yc);
            yuv_rgb32(pixmap2, Yc2);
            yuv_rgb32(pixmap2, Yc2);
        pixmap+=cois+cols-width;
        pixmap2+=cols+cols-width:
Yc+=width:
```

```
Engineering:KlicsCode:CompPict:Colour.c
```

```
Yc2+#width:
    )
١
*define rgb32_yuv(pixmap,Yc) \
    pixel.pixel=0x808080^*pixmap++; \
    r=pixel.rgb(1): \
    g=pixel.rgb(2): \
    b=pixel.rgb(3); \
    Y= (table[0xFF&r].ry + (g<<2)-table(0xFF&g].ry-table(0xFF&g).by + table(0xFF&b
    'Yc++ = limit(Y,16-128,235-128); \
U+= (r<<1) -g -table(0xFF4g).rv - table(0xFF4b).bu; \
    V+= (b<<1) -g -table(0xFF&r).rv - table(0xFF&g).bu;
         RGB32YUV(RGB_Tab *table.long *pixmap. short *Yc, short *Uc, short *Vc, int
void
              *pixmap2=pixmap+cols, *row, *end=pixmap+area;
     long
              *Yc2=Yc+width;
     short
     while(pixmap<end) (
         row=pixmap+width;
         while(pixmap<row) (
              Pixel pixel:
                       r,g,b,Y,U=0,V=0;
              long
              rgb32_yuv(pixmap,Yc);*/
              pixel.pixel=0x808080^*pixmap++;
              r=pixel.rgb[1];
              g=pixel.rgb[2];
              b=pixel.rgb(3);
              Y= (table[0xFF&r].ry + (g<<2)-table[0xFF&g].ry-table[0xFF&g].by + tabl
              r= (cable(uxff&r).ry + (g<<4)-table(uxff&g).ry-table
*Yc++ = limit(Y,16-128,235-128);
U+= (r<<1) -g -table[0xff&g].rv - table(0xff&b].bu;
V+= (b<<1) -g -table(0xff&r).rv - table(0xff&g).bu;</pre>
              rgb32_yuv (pixmap, Yc);
               rgb32_yuv(pismap2,Yc2);
               rgb32_yuv(pixmap2,Yc2);
               U>>=2;
               V>>=2:
               *Uc++=limit(U.16-128,240-128);
               *Vc++=limit(V,16-128,240-128);
          pixmap+=cols+cols-width;
          pixmap2+=cols+cols-width:
          Yc+=width;
          Yc2+=width;
     )
 }
 #define yuv_rgb32x2(pixmap,Y) \
     pixel.rgb(1)=over(Y+r); \
pixel.rgb(2)=over(Y+g); \
     pixel.rgb(3) = over(Y+b);
     pixmap(cols)=pixel.pixel; \
      pixmap++=pixel.pixel;
          YUV2RGB32x2(UV32_Tab *table.long *pixmap, short *Yc, short *Uc, short *Vc,
 void
               *piumap2=piumap+2*cols, *row, *end=piumap+area;
      long
               *Yc2=Yc+width:
      short
```

)

```
while(pixmap<end) (
               Yold=*Yc>>2, Yold2=*Yc2>>2;
       long
       row=pixmap+width*2;
       while(pixmap<row) (
           Pixel pixel:
           long
                   r.g.b.Y.U.V;
           U=0x00FF&((*Uc++)>>2);
           V=0x00FF&((*Vc++)>>2);
           r=table(U).ru;
           g=table(U).gu+table(V).gv;
           b=table(V).bv;
           Y=("YC++)>>2;
            Yold=(Y+Yold)>>1;
           yuv_rgb32x2(pixmap, Yold);
           Yold=Y:
           yuv_rgb32x2(pixmap,Yold);
            Y=(*YC++)>>2;
            Yold=(Y+Yold)>>1;
           yuv_rgb32x2(pixmap, Yold);
            Yold=Y:
           yuv_rgb32x2(pixmap, Yold);
            Y=(*YC2++)>>2;
            Yold2=(Y+Yold2)>>1;
           yuv_rgb32x2(pixmap2,Yold2);
            Yold2=Y:
           yuv_rgb32x2(pixmap2,Yold2);
           Y=(*Yc2++)>>2;
Yold2=(Y+Yold2)>>1;
           yuv_rgb32x2(pixmap2,Yold2);
           Yold2=Y:
           yuv_rgb32x2(pixmap2, Yold2);
       pixmap+=4*cols-2*width:
       pixmap2+=4*cols-2*width:
        Yc+=width:
        Yc2+=width;
   )
#define yuv_rgb8(pixel.Yc,index,dith) \
   Y=*YC++; \
   Y<<=3; \
   Y&= 0x3F00; \
   YI= U; \
   pixel.rgb(index)=table(Y).rgb(dith);
       YUV2RGB8(Pixel *table,long *pixmap, short *Yc, short *Uc, short *Vc, int a
            *pixmap2*pixmap+cols/1, *row, *end=pixmap+area/4;
   long
   short
            *Yc2=Yc+width;
   while(pixmap<end) (
```

```
Engineering:KlicsCode:CompPict:Colour.c
```

```
row=pixmap==idth/4;
         while(pixmap<row) (
              Pixel pixel, pixel2; long Y,U,V;
              U=*UC++;
              V= *VC++;
              U>>=2;
              V>>=6:
              U= (U&0xF0) | (V&0x0F);
              Yuv_rgb8(pixel,Yc,0,3);
              Yuv_rgb8(pixel,Yc,1.0);
Yuv_rgb8(pixel2,Yc2.0,1);
              yuv_rgb8(pixel2,Yc2,1,2);
              U=*UC++;
              V=*VC++;
              U>>=2;
              V>>=6:
              U= (U&0xP0). | (V&0x0F);.
              yuv_rgb8(pixel,Yc,2,3);
              yuv_rgb8(pixel,Yc,3,0);
yuv_rgb8(pixel2,Yc2,2,1);
              yuv_rgb8 (pixel2, Yc2, 3, 2);
              *pixmap++=pixel.pixel;
*pixmap2++=pixel2.pixel;
         pixmap+=(cols+cols-width)/4;
         pixmap2+=(cols+cols-width)/4:
         Yc+=width;
         Yc2+=width;
    )
)
#define yuv_rgb8x2(pixel.pixel2,Y,index,dith,dith2) \
    Y&= 0x3F00; \
    Y1= U; \
    pixel.rgb[index]=table(Y].rgb[dith]; \
pixel2.rgb[index]=table(Y].rgb[dith2];
void
         YUV2RGB8x2(Pixel *table,long *pixmap, short *Yc, short *Uc, short *Vc, int
             *pixmap2=pixmap+cols/2, *row, *end=pixmap+area/4;
*Yc2=Yc+width;
    long
    short
    while(pixmap<end) (
                  Yold=*Yc<<3, Yold2=*Yc2<<3;
         long
         row=pixmap+width/2;
         while(pixmap<row) (
              Pixel pixel, pixel2, pixel3, pixel4; long Y,U,V;
              U=*UC++;
              V=*VC++;
              U>>=2;
              V>>=6;
              U= (U£0x00F0) ( (V£0x000F);
             Y=(*YC++)<<3;
```

```
Yold=(Y+Told)>>1:
             yuv_rgb8x2(pixel.pixel2.Y.0.3.1);
             Yold=Y:
             yuv_rgb8x2(pixel.pixel2,Y,1,0,2);
             Yold=Y;
             Y=(*YC++)<<3;
             Yold=(Y+Yold)>>1;
             yuv_rgb8x2(pixel,pixel2,Y,2,3,1);
             Yold=Y:
             yuv_rgb8x2(pixel,pixel2,Y,3,0,2);
             Yold=Y:
             Y=(*Yc2++)<<3;
             Yold2=(Y+Yold2)>>1:
             yuv_rgb8x2(pixel3,pixel4,Y.0.3.1);
             Yold2=Y:
             yuv_rgb8x2(pixel3,pixel4,Y,1,0,2);
Yold2=Y;
             Y=(*Yc2++)<<3;
             Yo1d2 = (Y+Yo1d2) >>1;
             yuv_cgb8x2(pixel3,pixel4,Y,2,3,1);
             yuv_rgb8x2(pixel3,pixel4,Y,3,0,2);
Yold2=Y;
             pixmap(cols/4)=pixel2.pixel;
             *pixmap++=pixel.pixel;
             pixmap2(cols/4)=pixel4.pixel;
             *pixmap2++=pixel3.pixel:
        pixmap+=(cols+cols-width)/2;
        pixmap2+=(cols+cols-width)/2;
         Yc+=width;
        Yc2+=width;
)
*define yuv_rgbTEST(pixel.index,Y) \
    rgb_col.red=(Y+r<<8); \</pre>
    rgb_col.green=(Y+g<<8); \
rgb_col.blue=(Y+b<<8); \</pre>
    pixel.rgb(index)=Color2Index(&rgb_col);
        YUV2RGBTEST(UV32_Tab *table,long *pixmap, short *Yc. short *Uc, short *Vc.
void
             *pixmap2=pixmap+cols/2, *row, *end=pixmap+area/4;
    long
             *Yc2=Yc+width;
    short
    while(pixmap<end) {
                 Yold="Yc<<3, Yold2="Yc2<<3;
         long
        row=pixmap+width/2;
        while(pixmap<row) (
             RGBColor
                         rgb_col;
             Pixel pixel, pixel2;
```

```
long = r,g,b,Y,U,V;
    U=0x00FF&((*Uc++)>>2);
    V=0x00FF&((*VC++)>>2);
    r=table(U).ru;
    g=table(U).gu+table(V).gv;
b=table(V).bv;
    Y=(*YC++)>>2;
    Yold=(Y+Yold)>>1;
    rgb_col.red=(Yold+r<<8);</pre>
    rgb_col.green=(Yold+g<<8);</pre>
    rgb_col.blue=(Yold+b<<8);
    pixel.rgb(0)=Color2Index(&rgb_col);
    Yold=Y:
    yuv_rgbTEST(pixel,1,Yold);
    Y=(*YC++)>>2;
    Yold=(Y+Yold)>>1;
    yuv_rgbTEST(pixel,2,Yold);
    yuv_rgbTEST(pixel, 3, Yold);
    Y=(*Yc2++1>>2;
    Yold2=(Y+Yold2)>>1;
    yuv_rgbTEST(pixel2,0,Yold2);
    Yold2=Y:
    yuv_rgbTEST(pixel2,1,Yold2);
    Y=(*YC2++)>>2;
    Yold2=(Y+Yold2)>>1;
    yuv_rgbTEST (pixel2, 2, Yold2);
    Yold2=Y;
   yuv_rgbTEST(pixel2, 3, Yold2);
    pixmap(cols/4)=pixel.pixel;
    *pixmap++=pixel.pixel;
    pixmap2{cols/4}=pixel2.pixel;
*pixmap2++=pixel2.pixel;
pixmap+=(cols+cols-width)/2;
pixmap2+=(cols+cols-width)/2;
Yc+=width:
Yc2+=width;
```

```
© Copyright 1993 KLICS Limited
   All rights reserved.
   Written by: Adrian Lewis
68030 Colour space conversions
       machine mc68030
               'klics'
       seg
      include 'Traps.a'
       macro
     · DPY32x2
                   £ARGB, £row, £00, £01, £n0, £n1
       add.l
                   &n0,400.
       lsr.l
                   $1,€00
                                          ; interpolate first pixel
       add.1
                   &n1,&o1
       lsr.1
                   $1,&o1
                                          ; interpolate first pixel
       move.1
                   &o0, (&ARGB)
                   &row, &ARGB
&o0, (&ARGB)
       add.l
       move.l
       add.l
                   Erow, EARGB
       move.l
                   &ol. (&ARGB)
       add.1 .
                   Erow, EARGB
       move.1
                   GOL (GARGE)+
                   Enl.(EARGB)
       move.l
       sub.1
                   &row, &ARGB
                   &nl.(&ARGB)
       move. 1
       sub.1
                   &row, &ARGB
       move.l
                   ano, (&ARGB)
       sub.1
                   &row, &ARGB
       move.1
                   &nO, (&ARGB)+
       macro
       DPY32
                  EARGB, Erow, EoO, Eol, EnO, Enl
       move.1
                  £00, (£ARGB)
       add.1
                   &row. &ARGB
       move.l
                  401. (&ARGB) -
       move.l
                  Enl, (EARGB)
       sub.1
                   &row, &ARGB
       move.1
                   &n0, (LARGE)+
       macro
       UV2RGB32
                  EAU, EAV, ETAB
       add.l
                  $2048, &TAB
                                         ; move to uvtab
       move.w
                  £AU, d1
                                         ; Load U
       lsr.w
                   #2,d1
                  #$01FF.d1
       and. w
```

move.1

```
move.1
                    (&TAB, d1.w*8), d0
                                               ; UV now rg (u)
                                               ; Load V
                    £AV.dl
       move.w
       lsr.w
                    #2,d1
                    #$01FF.d1
       and.w
                    4(&TAB, d1.w*8), d0
                                               ; UV now rgb
       add.l
       move.l
                    d0.d1
                                               : 3 copies
                    d0,d2
       move.1
                    d0.d3
       move.1
                    #2048, &TAB
                                               ; restore ytab
       sub. l
       endm
       macro
       GETY32
                    LAY, LTAB, LRCBO, LRGB1
                    EAY. d4
       move.1
                     #2,d4
        lsr.w
        and.w
                     #$01FF.d4
       add. 1
                     (&TAB, d4.w*4), &RGB1
                                               ; RGB1+=YYY
        swap
                     d4
                     #2.d4
        lsr.w
                     #$01PF,d4
        and.w
        add.l
                     (4TAB, d4.w*4), 6RGB0
                                               ; RGBO+=YYY
        endm
       DACIO
        OVER32
                     &RGB
                     LRGB. d4
                                      ; copy pixel
; was it this rgb
        move.1
                     #$01010100,d4
        andi.l
                                       ; if not then quit
        beq.s
                     @nx_rgb
        btst.
                     #24,d4
                                      ; R overflow?
                     @bit16
                                      ; if not then continue
        beg. s
                     #23, &RGB
                                      : test sign
        btst
                     9pos23
                                      ; if positive
        beq.s
                     #$0000ffff, &RGB; underflow sets R to 0
        andi.1
                                      ; do next bit
                     @bit16
        bra.s
                     #S00ff0000, ERGB : overflow sets R to 255
@pos23
        ori.l
                                        G overflow?
                     +16.d4
@bit16
        btst
                                       ; if not then continue
                     @bit8
        beq.s
                     *15, &RGB
                                       ; test sign
        btst
                     epos16
                                       ; if positive
        beq.s
                                       ; underflow sets G to 0
                     #$00ff, &RGB
        andi.w
                                       ; do next bit
        bra.s
                     @bit8
                                       ; overflow sets G to 255
                     *Sff00.&RGB
8pos16
        ori.w
                     #8,d4
                                       ; B overflow?
Shit8
        btst
                                        if not then continue
                     end
        beq.s
                     #7, LRGB
                                       ; test sign
        btst
                                       ; under/over flow
                     LRGB
        sed
                     #$00fefefe, ERGB ; mask RGB ok
        andi.l
@end
Onx_rgb
        endn
        macro
                     EAH, &DO, &D1, &D2, &D3
        HASHOUT32
                     &D0.d4
```

```
Engineering:KlicsCode:CompPict:Colour.a
         add.l
                      £D1.d4
         add.1
                      &D2.d4
         add.l
                      &D3.d4
         andi.l
                      4$03e3e3e0.d4
         move.1
                      d4. &AH
         endm
         macro
         HASHCMP32
                      £AH, £D0, £D1, £D2, £D3
         move.l
                      &D0.d4
                      &D1.d4
&D2.d4
         add.l
         add.l
         add.1
                      &D3.d4
         andi.l
                      #$03e3e3e0,d4
         cmp.l
                      EAH, d4
         endm
OUT32X2 FUNC EXPORT
PS
         RECORD
                      8
table
         DS.L
pixmap DS.L
         DS.L
Ħ
         DS.L
         DS.L
width
         DS.L
height
         DS.L
                      1
rowByte DS.L
pixmap2 DS.L
                      1
         ENDR
LS
Y1
         RECORD
                      0, DECR
         DS.L
                      1
                                   : sizeof(short)*Yrow
                                                                      = 2°width
U_ex
         DS.L
                                   ; x end address
                                                                      = U+U_ix
                                   ; y end address
U_ey
         DS,L
                                                                      = U+width*height>>
U_ix
         DS.L
                                   : sizeof(short)*UVrow
                                                                     = width
                                                                  = 2*width
                                   ; sizeof(short)*Yrow
Y_Y
         DS.L
         DS.L
                                   : 4*rowBytes-sizeof(long)*Prow = 4*rowBytes-width
LSize
        EQU
         ENDR
        a0 - Y, a1 - U, a2 - V, a3 - pixmap, a4 - table, a5 - pixmap2
d0 - rgb00, d1 - rgb01, d2 - rgb10, d3 - rgb11, d4 - spare, d6 - old0, d7
        link
                      a6, *LS.LSize
                                                ; inc, width, fend and rowend are loca
        movem.l
                      d4-d7/a3-a5,-(a7)
                                                ; store registers
        move
                      SR.dO
                      PS.Y(a6),a0
                                                ; Y=YC
        move.l
                      PS.U(a6),a1
        move.l
                                                ; U=Uc
; V=Vc
                      PS.V(a6),a2
        move.l
                      PS.pixmap(a6),a3
PS.table(a6),a4
        move.1
                                                ; pm=pixmap
        move.1
                                                ; tab=table
        move.1
                      PS.pixmap2(a6),a5
                                                ; pm2=pixmap2
                      PS.width(a6),d0
        move.1
                                                ; LOAD width
                      d0.LS.U_ix(a6)
        move.1
                                                ; SAVE U_ix
```

PS.height(a6),dl

d0.d1

move.l

mulu.w

; LOAD height

: width height

FUNC

```
lsr.1
                      #1.d1
                                                 ; width*height/2
                      al, di
         add.l
                                                    U+width*height/2
        move.1
                      d1, LS. U_ey (a6)
                                                   SAVE U_ey
         add.l
                      0D.0D
                                                   width*2
                      d0, LS. Y1 (a6)
         move.1
                                                   SAVE YI
                      d0, LS.Y_y(a6)
         move.1
                                                 ; SAVE Y_Y
         1s1.1
                      #2.d0
                                                    width 8
         move.l
                      PS.rowByte(a6),d1
                                                 : LOAD rowBytes
         lsl.l
                      #2.d1
                                                   rowBytes*4
         sub. 1
                      d0.d1
                                                    rowBytes*4-width*8
        move.1
                      d1, LS. P_y (a6)
                                                 ; SAVE P_Y
        move.1
                      PS.rowByte(a6),d5
                                                : load rowBytes
        clr.1
                      d6
                                                ; clear old2
        clr.1
                      d7
                                                ; clear old1
@do_y
        move. 1
                      LS.U_ix(a6),d0
                                                : LOAD U_ixB
        add.1
                      a1,d0
                                                ; P+U_ixB
                      d0.LS.U_ex(a6)
        move.1
                                                ; SAVE U_exB
@do_x
        UV2RGB32
                      (a1)+, (a2)+, a4
                                                ; uv2rgb(*U++,*V++)
                      LS. Y1 (a6), d4
        move.l
                                                ; load Yrow ; add Yb to RGB values
        GETY32
                      (a0,d4.1),a4,d2,d3
        GETY32
                      (a0)+,a4,d0,d1
                                                ; add Ya to RGB values
        move.l
                      d0.d4
        or.l
                      di.d4
                      d2, d4
d3, d4
        or.l
        or.1
         andi.l
                      #$01010100,d4
        bne.s
                      Gover
                                                ; if overflow
@ok
        HASHOUT32
                      (a5)+,d0,d1,d2,d3
        DPY32x2
                      a3,d5,d6,d7,d0,d2
        DPY32x2
                     a3,d5,d0,d2,d1,d3
        move.1
                     d1,d6
                                                ; copy olds
        move.l
                     d3, d7
        cmpa.l
                     LS.U_ex(a6),a1
        blt.w
                     #do_x
        add.1
                     LS.Y_y(a6),a0
        add.1
                     LS. P_y (a6), a3
        cmpa.1
                     LS.U_ey(a6),a1
        blt.w
                     800_y
        movem.1
                      (a7)+,d4-d7/a3-a5
                                                : restore registers
        unlk
                     a6
                                                ; remove locals
        rts
                                                : return
        OVER32
Bover
                     d0
        OVER32
                     d1
        OVER32
                     d2
        OVER32
                     43
        bra
                     eok
        ENDFUNC
OUT32X2D
                     EXPORT
```

```
PS
         RECORD
                      8
table
        DS.L
pixmap
        DS.L
         DS.L
ï
11
         DS.L
         DS.L
width
         DS.L
                      1
height
        DS.L
rcwByte DS.L
pixmap2 DS.L
         ENDR
LS
         RECORD
                      0. DECR
Y1
         DS.L
                                   ; sizeof(short)*Yrow
                                                                       = 2*width
U_ex
         DS.L
                                                                       = U+U_ix
                                   ; x end address
U_ey
         DS.L
                                   ; y end address
                                                                       = U+width*height>>
U_ix
         DS.L
                                   ; sizeof(short)*UVrow
                                                                       - width
Y_Y
         DS.L
                                   ; sizeof(short) *Yrow
                                                                       = 2°width
P_y
         DS.L
                                   ; 4*rowBytes-sizeof(long)*Prow = 4*rowBytes-width
LSize
         EOU
         ENDR
         a0 - Y, a1 - U, a2 - V, a3 - pixmap, a4 - table, a5 - pixmap2
d0 - rgb00, d1 - rgb01, d2 - rgb10, d3 - rgb11, d4 - spare, d6 - old0, d7
                                                 ; inc, width, fend and rowend are loca
         link
                      a6.#LS.LSize
                      d4-d7/e3-a5,-(a7)
         movem. 1
                                                 ; store registers
                      PS,Y(a6),a0
PS.U(a6),a1
        move.l
                                                 ; Y=YC
        move.1
                                                  ひょりに
         move.1
                      PS.V(a5),a2
                                                ; V=Vc
        move.1
                      PS.pixmap(a6),a3
                                                ; pm=pixmap
         move.l
                      PS.table(a6),a4
                                                 ; cab=table
        move.1
                      PS.pixmap2(a6),a5
                                                ; pm2=pixmap2
        move.1
                      PS.width(a6),d0
                                                : LOAD width
                      d0,LS.U_1x(a6)
                                                  SAVE U_SX
        move.1
         move.1
                      PS.height(a6),dl
                                                  LOAD height
         mulu.w
                      d0.d1
                                                   width*height
         lsr.l
                      #1.dl
                                                   width*height/2-
         add.l
                      al.dl
                                                   U-width*height/2
         move.1
                      d1.LS.U_ey(a6)
                                                  SAVE U_ey
         add.1
                      d0.d0
                                                   width 2
                      d0.15.Y1(a6)
                                                  SAVE Y1
        move.l
                      d0.LS.Y_y(a6)
        move.1
                                                  SAVE Y_Y
                      #2,d0
                                                   widtl: 8
         1s1.1
                      PS.rowByre(a6),d1
                                                  LOAD rowbytes
        move.l
                      $2.d1
         1s1.1
                                                   rowPytes*4
                                                   rowbytes*4-width*8
                      40.41
         sub. l
        move.1
                      d1.LS.P_y(a6)
                                                ; SAVE P_Y
                      PS.rowbyte(a6),d5
        move.1
                                                ; load rowBytes
        clr.1
                      d6
                                                ; clear old2
        clr.1
                      47
                                                ; clear old1
@do_y
        move.l
                      LS.U_ix(a6),d0
                                                : LOAD U_ixB
        add.1
                      a1.d0
                                                  P+U_ixB
        move.1
                      d0, LS.U_ex(a6)
                                                : SAVE U_exB
@do_x
        UV2RGB32
                      (a1)+,(a2)+,a4
                                                ; uv2rgb(*U++,*V++)
        move. 1
                      LS.Yl(a6),d4
                                                ; load Yrow
                                                ; add Yb to RGB values
        GETY32
                      (a0,d4.1),a4,d2,d3
```

```
GETY32
                      (a0)+,a4,d0,d1
                                                 ; add YA to RGB values
         move.1
                      d0.d4
         or.l
                      d1,d4
                      d2,d4
         or.l
         or.l
                      45.66
         andi.l
                      $501010100,d4
         bne.w
                      Gover
                                                ; if overflow
@ok
         HASHCHP32
                      (a5)+,d0,d1,d2,d3
         bne.s
                      0diff
         add.1
                      #16.a3
                                                ; add four pixels
3cont
         move.1
                      d1.d6
                                                : copy olds
         move.1
                      d3, d7
         cmpa.1
                      LS.U_ex(a6),al
         blt.w
                      x_ob9
                      LS.Y_y(a6),a0
         add.1
         add.1
                      LS.P_y(a6),a3
         cmpa.1
                      LS.U_ey(a6),a1
         blt.w
                      8do_y
         movem. 1
                      (a7)+,d4-d7/a3-a5
                                                ; restore registers
         unlk
                      a6
                                                ; remove locals
         rts
                                                ; return
@diff
         move.1
                     d4, -4 (a5)
         DPY32x2
                     a3,d5,d6,d7,d0,d2
         DPY32x2
                      a3, d5, d0, d2, d1, d3
                     Scont
         bra.s
Gover
         OVER32
                     d0
        OVER32
                     di
         OVER32
                     <u>a2</u>
         OVER32
                     đ3
        bra
                     eok
        ENDFUNC
OUT32
        FUNC
                 EXPORT
PS.
        RECORD
table
        DS.L
        DS.L
pixmap
        DS.L
                     ī
        DS.L
                     1
        DS.L
                     1
width
                     1
        DS.L
height DS.L
                     1
rowByte DS.L
                     1
pixmap2 DS.L
        ENDR
LS
        RECORD
                     0, DECR
Yl
        DS.L
                                                                    = 2*width
                                  ; sizeof(short)*Yrow
U_ex
        DS.L
                     1
                                  ; x end address
                                                                    = U+U_ix
U_ey
        DS.L
                     1
                                  ; y end address
                                                                    = U+width*height>>
        DS.L
                                  ; sizeof(short)*UVrow
                     1
                                                                    = width
Y_Y
        DS.L
                     1
                                  ; sizeof(short)*Yrow
                                                                    = 2°width
P_y
LSize
        DS.L
                     1
                                  ; 2*rowBytes-sizeof(long)*Prow
                                                                   = 2*rowBytes-width
        EQU
```

```
ENDR
          a0 - Y, a1 - U, a2 - V, a3 - pixmap, a4 - table, a5 - pixmap2 d0 - rgb00, d1 - rgb01, d2 - rgb10, d3 - rgb11, d4 - spare, d6 - cld0, d7
                        a6. #LS.LSize
          link
                                                   : inc, width, fend and rowend are loca
          movem. 1
                        d4-d7/a3-a5,-(a7)
                                                 ; store registers
          move.1
                        PS.Y(a6), a0
                                                   ; Y=YC
          move.1
                        PS. U(a6), al
                                                   : U=Uc
          move.1
                        PS.V(a6),a2
                                                   ; V=Vc
          move.1
                        PS.pixmap(a6),a3
                                                   ; pm=pixmap
          move.1
                        PS.table(a6),a4
                                                   ; tab=table
          move.1
                        PS.pixmap2(a6),a5
                                                   : pm2=pixmap2
          move.1
                        PS.width(a6),d0
                                                   : LOAD width
                                                   : SAVE U_ix
          move.1
                        d0, LS. U_ix(a6)
                                                   ; LOAD height
          move.1
                        PS.height(a6),d1
                        d0,d1
          mulu.w
                                                      width*height
          lsr.l
                        *1.dl
                                                      width*height/2
          add.l
                        al.dl
                                                      U+width*height/2
                                                     SAVE U_ey width*2
          move.1
                        d1.LS.U_ey(a6)
          add.l
                        d0, d0
          move.l
                        d0.LS.Y1(a6)
                                                     SAVE YI
                        d0, LS.Y_y (a6)
          move.1
                                                     SAVE Y_Y
          add. 1
                        d0,d0
                                                      width 4
          move. 1
                        PS. rowByte(a6), d1
                                                   ; LOAD rowBytes
          add.l
                        dl.dl
                                                     rowBytes*2
          sub. 1
                        40,41
                                                      rowBytes*2-width*4
          move.l
                        d1.LS.P_y(a6)
                                                   ; SAVE P_Y
          move.l
                        PS.rowByte(a6),d5
                                                   ; load rowBytes
          move.1
                        LS. Y1 (a6), d6
                                                   : load Yrow
 لا_009
                        LS.U_ix(a6),d7
          move.1
                                                   ; LOAD U_ix8
          add. 1
                        al.d7
                                                   ; P+U_ixB
· @do_x
          UV2RGB32
                        (a1)+, (a2)+, a4
                                                   ; uv2rgb(*U++,*V++)
          GETY32
                        (a0,d6.1),a4,d2,d3
                                                  ; add Yb to RGB values ; add Ya to RGB values
                        (a0)+,a4,d0,d1
          GETY32
          move.1
                       d0,d4
                       d1.d4
          or.l
          or.1
                       d2,d4
          or.1
                       d3.d4
          andi.l
                       #$01010100,d4
          bne.s
                       @over
                                                   ; if overflow
 Pok
          HASHOUT32
                       (a5)+,d0,d1,d2,d3
          DPY32
                       a3.d5.d0.d2.d1.d3
          cmpa.1
                       d7.a1
          blt.w
                       edo_x
          add. 1
                       LS.Y_y(a6),a0
          add. 1
                       LS.P_y(a6),a3
                       LS.U_ey (a6), a1
          cmpa.1
          blt.w
                       690_7
          movem. 1
                      (a7)+.d4-d7/a3-a5
                                                  ; restore registers
```

```
unlk
                                                  ; remove locals
         rts
                                                  ; return
€over
         OVER32
                       dΩ
         OVER32
                       ď
         OVER32
                       d2
         OVER32
                       a3
         bra
                       Bok
         ENDFUNC
OUT32D FUNC
                  EXPORT
PS
         RECORD
                       8
table
         DS.L
                      1
pixmap DS.L
                      1
         DS.L
                      1
Ų
         DS.L
         DS.L
width
         DS.L
                      1
height DS.L
rowByte DS.L
pixmap2 DS.L
                      1
         ENDR
LS
         RECORD
                      0.DECR
Y1
        DS.L
                                    : sizeof(short) *Yrow
                                                                        = 2°vidth
U_ex
         DS.L
                                    ; x end address
                                                                        = U+U_ix
U_ey
        DS.L
                                    ; y end address
                                                                       = U+width*height>>
U_ix
         DS.L
                                   ; sizeof(short)*UVrow
; sizeof(short)*Yrow
                      1
                                                                       = width
Y_Y
        DS.L
                      1
                                                                       = 2°width
        DS.L
                      1
                                   ; 2°rowBytes-sizeof(long)*Prow = 2*rowBytes-width
LŠize
        EQU
        ENDR
        a0 - Y, a1 - U, a2 - V. a3 - pixmap, a4 - table, a5 - pixmap2
d0 - rgb00, d1 - rgb01, d2 - rgb10, d3 - rgb11, d4 - spare, d6 - Yrow, d7
        link.
                      a6, #LS. LSize
                                                ; inc, width, fend and rowend are loca
        movem.1
                     d4-d7/a3-a5,-(a7)
                                                ; store registers
        move.1
                     PS.Y(a6),a0
                                                ; Y=Yc
        move.1
                     PS. U(a6), al
                                                ; U=Uc
        move.1
                     PS. V(a6), a2
                                                ; V=Vc
        move.1
                     PS.pixmap(a6),a3
                                                ; pm=pixmap
        move. 1
                     PS.table(a6),a4
                                                ; tab=table
                     PS.pixmap2(a6),a5
        move.1
                                                : pm2=pixmap2
        move.1
                     PS.width(a6),d0
                                                ; LOAD width
                     d0.LS.U_ix(a6)
        move.1
                                                ; SAVZ U_ix
        move.1
                     PS.height(a6),d1
                                                : LOAD height
        mulu.w
                     d0.d1
                                                   width*height
        lsr.1
                     #1.dl
                                                   width*height/2
        add.1
                     al,d1
                                                  U+width*height/2
        move.l
                     d1, LS. U_ey (a6)
                                                : SAVE U_ey
        add.1
                     05,0b
                                                ; wideh*2
        move.l
                     d0, LS. Y1 (a5)
                                                 SAVE Y1
                     d0, LS.Y_y (a6)
        move.l
                                                ; SAVE Y_Y
        add.1
                     d0,d0
                                                  width*4
        move.l
                     PS.rowByte(a6),d1
                                                : LOAD rowBytes
        add.1
                     d1,d1
                                                : rowBytes*2
        sub.1
                     d0.d1
                                                  rowBytes*2-width*4
       move.1
                     d1, LS. P_y (a6)
                                               ; SAVE P_y
```

```
→ PS.rowByce(a6),d5
         move.l
                                                  ; load rowBytes
         move.1
                      LS. Y1 (a6) . d6
                                                  ; load Yrow
                       LS.U_ix(a5).d7
?do_y
         move.l
                                                  : LOAD U_1xB
                       a1, d7
         add.l
                                                  : P+U_ixB
3do_x
         UV2RGB32
                       (a1) + (a2) + a4
                                                  ; uv2rgb(*U++,*V++)
                       LS.Y1(a6), d4
         move.1
                                                  ; load Yrow
                       (a0,d6.1).a4,d2,d3
(a0)+.a4,d0,d1
         GETY32
                                                 ; add Yb to RGB values ; add Ya to RGB values
         GETY32
                      d0.d4
         move.1
         or.l
                      d1.d4
         or.l
                       d2.d4
         or.1
                      45,66
                       #$01010100.d4
         andi.l
         bne.s
                       @over
                                                  ; if overflow
@ok
         HASHCMP32
                      (a5)+.d0,d1,d2.d3
         bne.s
         addq
                       48,a3
                                                 ; add four pixels
3cont
         cmpa.1
                      d7.a1
                      6do_x
         blt.w
       - add.1
                      LS.Y_y(a6),a0
         add.1
                      LS.P_y(a6),a3
         cmpa.1
                      LS.U_ey (a6), a1
         blt.w
                      edo_y
         movem.1
                      (a7)+,d4-d7/a3-a5
                                                 ; restore registers
         unlk
                      a6
                                                 ; remove locals
         rts
                                                 ; return
                      d4,-4(a5)
a3,d5,d0,d2,d1,d3
@diff
         move. 1
         DPY32
                      Scont
         bra.s
Cover
         OVER32
                      đO
         OVER32
                      dl
         OVER32
                      d2
         OVER32
                      d3
        bra
                      Ook
         ENDFUNC
        macro
         UVOV
                      EVAL, EOV
        move.w
                      EVAL. LOV
         add.w
                      #$0200.£0V
         and.w
                      #SFC00, LOV
         beq.s
                      Gok
         SSC.W
                      FOA
         bge.s
                      Gpos
        move.w
                      #SOIFF, LVAL
         bra.s
                      Ook
                      #SFEOD, &VAL
<sub>epos</sub>
        move.w
Bok
         endm
```

```
UVLIMIT FUNC
UVLIMIT FUNC EXPORT * fix d0, d4, spare d1 d2
         σνον
                      d0,d1
         swap
                      d0
         UVOV
                      d0,d1
                      do
         swap
         UVOV
                      d4,d1
         swap
                      d4
         UVOV
                      d4.d1
        swap
                      d4
         rts
         ENDFUNC
         macro
         UVOVER
                      &U, &V
         move.l
                      #$02000200,d1
        move.1
                      d1,d2
         add.l
                      EU.dl
         add.l
                      &V, d2
         or.l
                      d2.d1
         andi.l
                      #SFC00FC00,d1
        beq.s
                      QUVok
         bsr
                      UVLIMIT
@UVok
         endm
        macro
        GETUV
                      EAU, EAV, ESP, EUV
                      (£AU)+,£SP
(£AV)+,£UV
        move.1
        move.1
        UVOVER
                      &SP. &UV
        lsr.1
                      +5.4UV
                      #$03e003eG,43P
        andi.l
        andi.1
                      #$001F001F, &UV
        or.1
                      &SP, &UV
                                                  ; UV==$00UV00UV
        swap
                      VU3
        endm
        macro
                      SAY, SIND, SUV, SRO, SR1
        GETY
                      &AY, &R1
#5, &R1
        move.1
                                                 ; (2+) Y=Y0Y1
        ls1.1
                                                 ; (4) Y=Y0XXY1XX
                      #SFC00FC00, &R1
        andi.l
        OF.W
                      &UV, &R1
                                                 ; (2) Y=Y1UV
                      (&IND, &R1 .w*4), &R0
        move.1
                                                 ; (2+) R0=0123 (Y1)
        SWAD
                                                 ; (4) Y=Y0XX
; (2) Y=Y0UV
                      £R1
        OF.W
                      &UV. &R1
        move.1
                      (&IND, &R1 .w*4), &R1
                                                 ; (2+) R1=0123 (YO)
        endm
        macro
        nv8
                      LAU, LAV, ESP, EUV
                     (£AU)+,&SP
        move.1
                     (£AV)+, £UV
£SP. £UV
        move.1
        UVOVER
```

```
lsr.l
                      #2. &SP
        lsr.l
                      #6, &UV
        andı.l
                      #$00F000F0.&SP
                      #$000F000F.&UV
        andı.l
                                                 ; UV==$00UV00UV
        or.1
                      SP. LUV
        swap
                      SUV
        enám
        macro
        Y2 TND
                      EY, EIND, EUV, ED0, ED1
        move.1
                      &Y, &DO
                                                 : d0=Y0Y1
                                                 ; d0=Y0XXY1XX
        151.1
                      #3, &DO
                      AUV. ADO
                                                 ; d0=Y0XXY1UV
        move.b
                      #S3FFF.&D0
                                                 ; d0=0YUV(1)
        andi.w
                                                 ; find clut entries
        move.1
                      (&IND, &D0 .w=4), &D1
                                                 XX0Y=0b;
         gwap
                      &D0
                      EUV. EDO
                                                . : d0=Y0UV
        move.b
                      #$3FFF.&D0
(&IND,&D0 .w=4),&D0
        andi.w
                                                 : d0=0YUV(0)
                                                 ; find clut entries
        move.1 ·
         endm
OUT8
        FUNC
                 EXPORT
PS
        RECORD
                      8
cable
         DS.L
                      ١
pixmap
        DS.L
        DS.L
U
        DS.L
٧
        DS.L
width
        DS.L
height
        DS.L
rowByte DS.L
pixmap2 DS.L
        ENDR
        RECORD
                      0. DECK
LS
                                                                       = 2*width
                                   ; sizeof(short)*Yrow
Yl
        DS.L
                                                                       = U+U_ix
U_ex
         DS.L
                                   ; x end address
                                                                       = U+width*height>>
        DS.L
U_ey
                                   ; y end address
                                   ; sizeof(short)*UVrow
                                                                       = width
U_ix
        DS.L
                      1
                                                                       = 2*width
                                   ; sizeof(short)*Yrow
Y_Y
        DS.L
                      1
                                   ; 2 rowBytes-sizeof(long)*Prow = 2*rowBytes-width
P_y
         DS.L
LSize
         EQU
         ENDR
        a0 - Y, a1 - U, a2 - V, a3 - pixmap, a4 - table, a5 - pixmap2
d0 - rgb00, d1 - rgb01, d2 - rgb10, d3 - rgb11, d4 - spare, d6 - old0, d7
                                                ; inc. width, fend and rowend are loca
                      a6, #LS.LSize
        link
                      d4-d7/a3-a5,-(a7)
                                                ; store registers
        movem.l
                                                ; Y=YC
        move.l
                      PS.Y(a6),a0
                                                ; U=UC
        move.l
                      PS.U(a6),al
        move.l
                      PS.V(a6),a2
                                                ; V=Vc
         move.l
                      PS.pixmap(a6),a3
                                                ; pm=pixmap
                      PS.table(a6),a4
                                                ; tab=table
        move.1
        adda.i
                      #$00020000,a4
                                                ; tab+=32768 (longs)
                      PS.pixmap2(a6),a5
                                                ; pm2=p1xmap2
        move.1
                      PS.width(a6).d0
                                                : LOAD width
        move.1
```

Y8x2

Engineering: KlicsCode: CompPict: Colour.a

```
d0.LS.U_ix(a6)
         move.1
                                                 : SAVE U_ix
        move.1
                      PS.height(a6),dl
                                                 : LOAD height
         mulu.w
                      d0,d1
                                                    width*height
         lsr.l
                      #1.d1
                                                    width*height/2
         add.l
                      al.dl
                                                    U+width*height/2
         move. 1
                      d1. LS. U_ey (a6)
                                                 ; SAVE U_ey
         move.1
                      PS.rowByte(a6),d1
                                                   LOAD rowBytes
         add.l
                      dl.dl
                                                    rowBytes 2
         sub.l
                      d0,d1
                                                    rowBytes*2-width
         move. 1
                      d1.LS.P_y(a6)
                                                 : SAVE P_y
         add.l
                      d0.d0
                                                    width*2
                      d0.LS.Y1(a6)
         move. 1
                                                 ; SAVE Y1
        move.1
                                                 ; SAVE Y_Y
                      d0, LS.Y_y(a6)
                      PS.rowByte(a6),d5
LS.Y1(a6),d6
        move. 1
                                                 ; load rowBytes
        move.1
                                                 ; load Yrow
9do_y
        move. 1
                      LS.U_ix(a6),d7
                                                 ; LOAD U_ixB
        add.l
                      al, d7
                                                 ; P+U_ixB
X_ob9
       GETUV
                      a1, a2, d0, d4
        GETY
                      (a0,d6.w),a4,d4,d2,d3
                                                 ; d2=X0XX. d3=XXlX
        GETY
                      (a0)+,a4,d4.d0,d1
                                                 ; d0=XXX0, d1=1XXX
        move.w
                      d3, d2
                                                ; d2=x01x
        1s1.1
                      #8, d2
                                                ; d2=01XX
        move.w
                     d0.d1
                                                ; d1=1xx0
        swap
                                                ; d1=x01x
                      #8, 41
        181.1
                                                ; d1=01XX
        swap
                     d4
                                                ; next UV
        GETY
                      (a0,d6.1),a4,d4,d0,d3
                                                ; d0=X2XX, d3=XX3X
        move.w
                     d3.d0
                                                  d0=x23x
                     #8.₫0
        lsr.1
                                                : d0=XX23
        move.w
                     d0,d2
                                                  d2=0123--
                     (a0)+,a4,d4,d0,d3
        GETY
                                                ; d0=xxx2, d3=3xxx
        mové.w
                     d0,d3
                                                  d3=3XXX2
        swap
                     aз
                                                  d3=X23X
        isr.1
                     #8,d3
                                                  d3=XX23
        move.w
                     d3,d1
                                                ; d1=C123
                     d2, (a3,d5) d1, (a3)+
        move.1
        move.1
        CMDa.1
                     d7, a1
        blt.w
                     @do_x
        add. 1
                     LS.Y_Y(26), a0
        add. 1
                     LS. P_y(a6), a3
        cmpa.1
                     LS.U_ey(a6),a1
        blt.w
                     @do_y
       movem. 1
                     (a7)+,d4-d7/a3-a5
                                               ; restore registers
        unlk
                     a6
                                               ; remove locals
        rts
                     ; return
        ENDFUNC
       MACTO
```

SAY, &IND, &UV, wold

```
move.1
                SAY. do
                                            ; (2+) Y=Y0Y1
  lsl.l
                #3.d0
                                             : (4) Y=Y0XXY1XX
  swap
                40
                                             : (4) Y=Y1XXY0XX
  add. w
                d0.&old
                                            : (2) old=old+Y0
  lsr.w
                *1.&old
                                              (4) old=(old+Y0)/2
  move.5
                AUV. Gold
                                             : (2) old=YIOUV
  andi.w
                *S?FFP.&old
                                            : (4) old=0YUV(I0)
  move.1
                (&IND,&old .w*4),d1
                                            : (2+) d1=X1X3
 move.w
                d0.&old
                                            : (2) old=Y0
: (2) Y=Y0UV
  move.b
                0D, VU3
  andi.w
                *S3FFF.d0
                                            ; (4) Y=0YUV(0)
 move.1
                (&IND, d0, w*4), d2
                                            ; (2+) d2=0X2X
 move.w
                d1.d3
                                            ; (2) exg.w dl.d2
 move.w
                d2.d1
                                            : (2) d1=X12X
 move.w
                d3,d2
                                            ; (2) d2=0xx3
  SWAD
                d2
                                            ; (4) d2=X30X
  151.1
                #8,d1
                                            ; (4) d1=12xx
  1s1.1
                #8.d2
                                            : (4) d2=30XX
  swap
                d0
                                           ; (4) Y=Y1XX
  add.w
                d0.&old
                                          ; (2) old=old+Y1
; (4) old=(old+Y1)/2
  lsr.w
                #1.&old
 move.b
                LUV. Lold
                                           ; (2) old=YILUV
  andi.w
                #$3FFF. &old
                                            ; (4) old=0YUV(I1)
 move.l
                (&IND.&old .w*4),d3
                                         · ; (2+) d3=%1%3
                d0. cold
 move.w
                                            ; (2) old=11
 move.b
                EUV, do
                                            ; (2) Y=YOUV
 andi.w
                #$3FFF.d0
                                            : (4) Y=0YUV(0)
                (£IMD, d0, w*4), d0
 move.1
                                            : (2+) d0=0x2x
                                           ; (2) exg.w d0,d3
; (2) d0=0xx3
 move.w
                d0,d1
 move.w
                d3, d0
 move.w
                41,43
                                           ; (2) d3=x12x
 swap
               d0
                                            ; (4) d0=X30X
  lsr.l
                #8,40
                                           ; (4) d0=xx30
 lsr.1
               #8,d3
                                           ; (4) d3=X12X
 move.w
                                           : (2) d2=3030 (YiY0YiY1) (1)
; (2) d1=2121 (YiY0YiY1) (2)
               d0,d2
 move. w
               d3.d1
 endm
 macro
 Y8x2a
               EAY, & IND, &UV
 GETY
               EAY, EIND, EUV, d1, d2
 move.1
               £AY,d2
                                           : (2+) Y=Y0Y1
               #3,d2
&UV,d2
 lsl.1
                                           ; (4) Y=Y0XXY1XX
; (2) Y=Y1UV
 move.b
 andi.w
               #S3FFF.d2
                                           ; (4) Y=0YUV(Y1)
 move.1
               (&IND, d2.w-4), d1
                                           ; (2+) dl=0123 (Y1)
                                           ; (4) Y=f0XX
; (2) Y=Y0UV
; (4) Y=0YUV(Y0)
 SWAD
               d2
move.b
               EUV. d2
 andi.w
               #$3FFF.d2
 move.l
               (&IND, d2.w*4), d2
                                           ; (2+) d2=0123 (Y0)
                                           ; (2) exg.w d2.d1
; (2) d1=0123 (Y1Y0)
; (2) d2=0123 (Y0Y1)
 move.w
               d1,d0
 move.w
               d2.d1
 move.w
               d0.d2
 swap
               dl
                                           ; (4) d1=2301 (Y0Y1)
 enda
 macro
 Y8x2b
               EAY, EIND, EUV
 GETY
               &AY, & IND, &UV, d1, d2
```

```
move.1
                      &AY, d2
                                                   : (2+) Y=Y0Y1
         1s1.1
                       #3.d2
                                                   : (4) Y=Y0XXY1XX
         move.b
                       £UV, d2
                                                   : (2) Y=Y1UV
         andi.w
                                                   : (4) Y=0YUV(Y1)
                       #$3FFF, d2
         move.1
                       (&IND, d2.w*4), d1
                                                   : (2+) d1=0123 (Y1)
         swap
                      d2
                                                   ; (4) Y=Y0XX
                       5UV, d2
         move.b
                                                  ; (2) Y=YOUV
         andi.w
                       #$3FFF.d2
                                                  : (4) Y=0YUV(YU)
         move.l
                       (&IND, d2.w*4).d2
                                                  ; (2+) d2=0123 (YO)
         ror.l
                       #8,d2
                                                  ; (6) d2=3012 (Y0)
         ror.1
                       #8,d1
                                                  ; (6) dl=3012 (Y1)
        move.w
                       dl.d0
                                                  ; (2) exg.w d2,d1
        move.w
                      d2.d1
                                                  ; (2) d1=3012 (Y1Y0)
         move.w
                      d0.d2
                                                  ; (2) d2=3012 (Y0Y1)
         Swap
                      dl
                                                  ; (4) d1=1230 (Y0Y1)
         ror.w
                       #8.dl
                                                 (6) d1=1203 (Y0Y1)
         endm
OUT8x2 FUNC
                  EXPORT
25
         RECORD
                      R
table
        DS.L
                      1
gamag
        DS.L
        DS.L
         DS.L
Ħ
        DS.L
        DS.L
width
height
        DS.L
rowByce DS.L
pixmap2 DS.L
                      1
        ENDR
LS
        RECORD
                      0. DECR
Y1
        DS.L
                                    ; sizeof(short)*Yrow
                                                                        = 2*width
U_ex
                                   ; x end address; y end address
        DS.L
                                                                        = U+U_{ix}
        DS.L
U_ey
                                                                        = U+width*height>>
U_ix
        DS.L
                                    : sizeof(short)*UVrow
                                                                        = width
Y_Y
        DS.L
                                    ; sizeof(short)*Yrow
                      1
                                                                        = 2*width
        DS.L
                                    ; 4*rowBytes-sizeof(long)*Frow = 4*rowBytes-width
LSize
        EQU
        ENDR
        a0 - Y, a1 - U, a2 - V, a3 - pixmap, a4 - table, a5 - pixmap2
d0 - rgb00, d1 - rgb01, d2 - rgb10, d3 - rgb11, d4 - spare, d5 - old0, d7
        link
                                                 ; inc, width, fend and rowend are loca; store registers
                      a6, #LS.LSize
        movem.l
                      d4-d7/a3-a5, -(a7)
        move. 1
                      PS.Y(a6),a0
                                                 ; Y=YC
        move.l
                      PS.U(a6),a1
                                                 ; U=Uc
        move. 1
                      PS.V(a6),a2
                                                 ; Y=Vc
        move.l
                      PS.pixmap(a6),a3
                                                 ; pm=pixmap
        move.1
                      PS.table(a6),a4
                                                 ; tab=table
        adda.1
                      #$00020000,a4
                                                 ; tab+=32768 (longs)
        move.1
                      PS.pixmap2(a6), a5
                                                 ; pm2=pixmap2
                                                 ; LOAD width ; SAVE U_ix
        move.l
                      PS.width(a6),d0
        move.1
                      d0, LS. U_ix(a6)
                                                 ; LOAD height
        move.1
                      PS.height(a6),d1
        mulu.w
                      d0, d1
                                                    width*height
        lsr.1
                      #1.d1
                                                    width*height/2
```

```
al.dl
dl.LS.U_ey(a6)
        add.l
                                                  U-width height/2
        move.1
                                                : SAVE U_ey
        add.l
                     0b.0b
                                               ; width*2
                                                . SAVE Y1
        move.1
                     d0.LS.Y1(a6)
                     d0. LS. Y_Y(a6)
        move.1
                                               : SAVE Y_Y
        move.1
                     PS.rowByte(a6),dl
                                               : LCAD rowBytes
                                               : rowByces*2
: rowByces*4
        add.l
                     d1.d1
        add.l
                     d1.d1
        sub.1
                     d0.d1
                                                : rowBytes*4-width*2
                     d1.15.P_y(a6)
        move.1
                                               ; SAVE P_Y
        move.1
                     PS.rowByte(a6),d5
                                               ; load rowBytes
        clr.1
                     d6
        clr.ī
                     d7
3do_y
        move.l
                     LS.U_ix(a6),d0
                                               ; LOAD U_ixB
        add.l
                     a1.d0
                                               ; P+U_ixB
       . move.1
                     d0, LS.U_ex(a6)
                                               ; SAVE U_exB
edo_x
        GETUV
                     a1.a2,d0.d4
                                               ; d4=00UV00UV (10)
        Y8x2a
                      (a0),a4,d4;,d6
                                               ; calc d2.dl pixels
                     d2, (a3)
        move.1
        add.1
                     d5.a3
        move.1
                     d1.(a3)
        add.l
                     d5, a3
        move.1
                     LS.Y1(a6), d0
                                               ; load Yrow
                                               ; calc d2,d1 pixels
        Y8x2b
                     (a0,d0.w),a4,d4;,d7
                     d2, (a3)
d5, a3
        move.1
        add.1
        move.l
                     d1.(a3) +
        SWAD
                                               ; next UV
        acidq.1
                     #4,a0
                                               ; next Ys
        move.1
                     LS.Y1 (a6), d0
                                               ; load Yrow
        Y8x2b
                     (a0,d0.w).a4,d4;,d7
                                              ; calc d2,d1 pixels
        move. Y
                     d1, (a3)
        sub.l
                     d5, a3
        move.1
                     d2, (a3)
        sub.l
                     d5, a3
        Y8x2a
                     (a0)+,a4,d4;,d6
                     d1.(a3)
        move.1
        sub.1
                     d5.a3
                     d2, (a3)+
        move.1
                     LS.U_ex(a6),a1
        cmpa.l
        blc.w
                     ago_x
        add.1
                     LS.Y_Y(a6),a0
        add.l
                     LS.P_y(a6),a3
        cmpa.1
                     LS.U_ey (a6), a1
                     edo_y
        blt.w
                     (a7)+,d4-d7/a3-a5
        movem. 1
                                               ; restore registers
        unlk
                     a6
                                               ; remove locals
                     ; return
        rts
        ENDFUNC
```

- 747 -

```
MACTO
         RGB2Y
                       ERGB, LY, LU, LV, LAY
         move.1
                       ERGB, d2
                                                  ; pixel='pixmap
         ecri.1
                       #$808080,d2
                                                   pixel^=0x808080
         clr.w
                       d1
                                                   B=0
         move.b
                       d2.q1
                                                  ; B=pixel(3)
         move. 1
                       4 (a4, d1, w*8), d0
                                                   d0=by.bu
         sub.w
                       ₫0,&U
                                                  ; U-=bu
         swap
                       do
                                                   d0=bu, by
         move.w
                      d0, &Y
                                                 ; Y=by
         ext.w
                       d1
                                                  ; (short)B
         add.w
                      dl,dl
                                                  ; B*=2
         add.w
                      d1, &V
                                                   V+=B<<1
         1sr.1
                       #8,d2
                                                 ; pixel>>=8
         clr.w
                      d1
                                                   G=0
         move.b
                      d2, d1
                                                   Gapixel[3].
         move. 1
                       (a4,d1.w*8),d0
                                                   d0=gry,gv
         sub. w
                      d0,£Ŭ
                                                   U-zgv
                                                   d0=gv,gry
         swap
                      dО
         sub. w
                      d0,&Y
                                                   Y-=gry
                      4(a4,d1.w*8),d0
         move.1
                                                   d0=gby, gu
         sub.w
                      d0, &v
                                                   V-=gv
         swap
                      d0
                                                   d0=gu,gby
         sub.w
                      dO, &Y
                                                   Y-=gby
         ext.w
                      dl
                                                   (short)G
                      d1,£U
         sub.w
                                                   U-=g
         sub.w
                                                   V-=g
                      dl. EV
         lsl.w
                      #2,d1
                                                   G<<=2
         add.w
                      di, ay
                                                   Y+=B<<1
         lsr.1
                      #B, d2
                                                   pixel>>=8
         move.1
                      (a4.d2.w*8),d0
                                                   d0=ry,rv
         sub.w
                      dO, AV
                                                   V-srv
         swap
                      40
                                                   d0=rv.ry
         add.w
                      do, EY
                                                   Y+=EY
                      d2
d2,d2
         ext.w
                                                   (short)R
         add.w
                                                   R*=2
         add.w
                      42,60
                                                   U+=R<<2
                      *SFE40.&Y
         CMD1.W
                                                   Y>=-448
         bge.s
                      eok
                                                   if greater
        move.w
                      #SFE40,&Y
                                                   Y = -448
        bra.s
                      end
                                                   save
Cok
         cmpi.w
                      #$01C0,&Y
                                                  Y< 448
if less
        blt.s
                      0end
                                                :
        move.w
                      #$01C0,&Y
                                                  Y= 463
@end
        move.w
                      EY, EAY
                                                 ; Save Y
         endm
IN32
        FUNC
                 EXPORT
PS
        RECORD
table
        DS.L
pixmap
        DS.L
        DS.L
IJ
        DS.L
v
        DS.L
width
        DS.L
height
        DS.L
rowByte
        DS.L
        ENDR
LS
        RECORD
                      0.DECR
```

1

```
Υ1
         DS.L
                    =1
                                    ; sizeof(short) *Yrow
                                                                        = 2 width
U_ex
         DS.L
                                                                        = U+U_ix
                      1
                                    ; x end address
U_ey
         DS.L
                                    ; y end address
                                                                        = U+width*height>>
)_:x
Y_Y
        DS.L
DS.L
                      1
                                    ; :1zeof(short)*UVrow
                                                                        = width
                                   ; sizeof(short) 'Yrow
                                                                        = 2°width
         DS.L
                                    : 2*rowBytes-sizeof(long)*Prow = 2*rowBytes-width
LSize
         EQU
        EP!DR
        a0 - Y. al - U. a2 - V. a3 - pixmap, a4 - table, a5 - pixmap2
d0 - rgb00, d1 - rgb01, d2 - rgb10, d3 - rgb11, d4 - spare, d6 - old0, d7
         link
                      a6, #LS.LSize
                                                 ; inc, width, fend and rowend are loca
        movem.l
                      d4-d7/a3-a5, -(a7)
                                                 ; store registers
        move.1
                      PS.Y(a6),a0
                                                 ; Y=YC
        move.1
                      PS.U(a6),al
                                                 : U=Uc
        move.1
                      PS. V(a6), a2
                                                 ; V=Vc
        move.1
                      PS.pixmap(a6),a3
                                                 : pm=pixmap
        move.1
                      PS.table(a6),a4
                                                 ; tab=table
        move.1
                      PS.width(a6).d0
                                                 ; LOAD width
                                                : SAVE U_ix
: LOAD height
        move.1
                      d0, LS. U_ix(a6)
        move.1
                      PS.height(a5),dl
        mulu.w
                      d0,d1
                                                   width*neight
                      #1,d1
         lsr.l
                                                    width*height/2
        add.l
                      al,d1
                                                   U+width*height/2
        move.1
                                                ; SAVE U_ey ; width*2
                      dl. LS. U_ey(a6)
        add.l
                      d0.d0
                                                SAVE Y1
        move.l
                      d0.LS.Y1(a6)
                      d0, LS. Y_Y(a6)
        move.1
        add.l
                      d0,d0
                                                    width*4
        move.1
                      PS.rowByte(a6),dl
                                                ; LOAD rowBytes
        add.1
                      d1,d1
                                                  rowBytes*2
        sub.1
                      40.41
                                                    rowBytes*2-width*4
        move.l
                                                : SAVE P_y
                      d1, LS. P_v(a6)
        move.1
                      PS.rowByte(a6),d7
                                                ; load rowBytes
        move.1
                      LS.Y1 (a6), d6
                                                ; load Y1
        move.1
                      LS.U_ix(a6),d0
940_A
                                                ; LOAD U_ixB
        add.1
                      a1, d0
                                                   P+U_ixB
        move. 1
                      d0, LS. U_ex(a6)
                                                : SAVE U_exB
%_ob9
        clr.w
                      d4
                                                ; U=0
        clr.w
                                                : V=0
        RGB2Y
                      (a3,d7.w),d3,d4.d5,(a0,d6.w); Convert pixel
        RGB2Y
                      (a3)+,d3,d4,d5,(a0)+
                                                    ; Convert pixel
        RGB2Y
                      (a3,d7.w),d3.d4.d5,(a0,d6.w); Convert pixel
        RGB2Y
                      (a3)+,d3,d4,d5,(a0)+
                                                   ; Convert pixel
        asr.w
                      #2.d4
                                                ; U>>=2
        asr.w
                      #2,d5
                                                ; V>>=2
        cmpi.w
                     *SPE40.d4
                                                ; U>=-448
        bge.s
                     Boku
                                                ; if greater
        move.w
                      SPE40.de
                                                ; U= -448
        bra.s
                     9doV
                                                ; save
                     #$01C0.d4
                                               ; U< 448
; if less
eoku
        CEROI. W
        blt.s
                     Vobs
        move. w
                     #501C0.d4
                                                ; U= 448
```

DS.L

1

```
#SFE40.d5
Sqov
         стрі. ч 🗻
                                                 ; V>=-448
                      gokV
         bge.s
                                                 ; if greater
         move. w
                      #SFE40.d5
                                                 : V = -448
         bra.s
                      eend
                                                 : save
                      #$01CC.d5
BOKY
         cmpi.w
                                                 : V< 448
         blc.s
                      @end
                                                 ; if less
         move.w
                      #$01C0.d5
                                                 ; V= 448
@end
         nove. w
                      d4.(a1)+
                                                 ; Save U
         move.w
                      d5, (a2) +
                                                 ; Save V
         cmpa.l
                      LS.U_ex(a6),a1
         blt.w
                      x_ob9
         add.1
                      LS.Y_y(a6),a0
         add.1
                      LS.P_y(a6),a3
         cmpa.1
                      LS.U_ey(a6),a1
         blt.w
                      @do_y
         movem.1
                      (a7)+,d4-d7/a3-a5
                                                ; restore registers
         unlk
                      a6
                                                ; remove locals
         rts
                      : return
         ENDFUNC
                     ----
         macro
         UV:16
                      EAU, EAV, ESP, EUV
         move.l
                      (&AU)+, &SP
         move.1
                      (&AV)+, &UV
                      45,4UV
         UVOVER
         lsr.1
        andi.l
                      *$03e003e0,&SP
        andi.l
                      $$001F001F,&UV
         or.l
                      ESP, EUV
                                                ; UV==$00UV00UV
         swap
                      FUV
         endm
        macro
         Y16x2
                     SAY, SIND, SUV
        move.1
                     &AY, d2
                                                ; (2+) Y=Y0Y1
        151.1
                     #5,d2
#$PC00FC00,d2
                                                ; (4) Y=Y0XXY1XX
        andi.l
        or.w
                     EUV, d2
                                                ; (2) Y=Y1UV
                      (&IND, d2.w*4), d1
        move.1
                                                ; (2+) d1=0123 (Y1)
                                               ; (4) Y=Y0XX
; (2) Y=Y0UV
; (2+) d2=0123 (Y0)
        SWAD
                     · d2
                     £UV, d2
        or.w
        move.1
                     (&IND, d2. w*4), d2
        endn
OUT16x2 FUNC
                 EXPORT
PS
        RECORD
table
        DS.L
                     1
        DS.L
pixmap
Y
                     1
        DS.L
                     1
IJ
        DS.L
                     1
```

```
Engineering:KlicsCode:CompPict:Colour.a
```

```
width
         DS.L
height DS.L
rowByte DS.L
                       1
pixmap2 DS.L
         ENDR
LS
         RECORD
                      0.DECR
Y1
         DS.L
                                                                        = 2°width
                                   ; sizeof(short)*Yrow
U_ex
         DS . L
                                    : x end address
                                                                        = U+U_ix
U_ey
U_ix
         DS.L
                                    ; y end address
                                                                        - U+width*height>>
         DS.L
                      1
                                    ; sizeof(short)*UVrow
                                                                        = width
٧_̈У
         DS.L
                      1
                                   : sizeof(short)*Yrow
                                                                        = 2°width
P_y
         DS.L
                                    : 4*rowBytes-sizeof(long)*Prow = 4*rowBytes-width
LSize
         EQU
         ENDR
         a0 - Y. a1 - U. a2 - V. a3 - pixmap, a4 - table, a5 - pixmap2 d0 - rgb00, d1 - rgb01, d2 - rgb10, d3 - rgb11, d4 - spare, d6 - old0, d7
         link
                      a6.#LS.LSize
                                                 ; inc, width, fend and rowend are loca
         movem. 1
                      d4-d7/a3-a5, -(a7)
                                                 ; store registers
         move.1
                      PS.Y(a6),a0
                                                 : Y=Yc
         move.1
                      PS.U(a6),a1
                                                 : U=Uc
         move.1
                      PS.V(a6),a2
                                                 : V=Vc
         move.1
                      PS.pixmap(a6),a3
                                                 ; pm=pixmap
         move.1
                      PS.table(a6),a4
                                                 ; tab=table
         adda.l
                      #$00020000,a4
                                                 ; tab+=32768 (longs)
         move.1
                      PS.pixmap2(a6),a5
                                                 : pm2=pixmap2
                                                ; LOAD width ; SAVE U_ix
         move.l
                      PS.width(a6),d0
         move.l
                      d0.LS.U_ix(a6)
         move.1
                      PS.height(a6),d1
                                                 ; LOAD height
         mulu.w
                      d0,d1
                                                    width height
                      #1,d1
         lsr.1
                                                    width*height/2
         add.1
                      al,dl
                                                    U-width*height/2
         move.l
                      d1, LS. U_ey (a6)
                                                 ; SAVE U_ey
         add.1 .
                      d0.d0
                                                    width*2
         move.1
                      d0, LS. Y1 (26)
                                                   SAVE Y1
        move.l
                      d0, LS. Y_y (a6)
                                                   SAVE Y_y width'4
         add.1
                      d0,d0
        move.1
                      PS.rowByte(a6),d1
                                                ; LOAD rowBytes
        add.l
                      d1,d1
                                                   rowBytes*2
         add.1
                      dl.dl
                                                    rowBytes*4
        sub.1
                      d0.d1
                                                    rowBytes*4-width*4
        move.1
                      d1.LS.P_y(a6)
                                                : SAVE F_Y
                      PS.rowByte(a6).d5
        move.l
                                                ; load rowBytes
        clr.1
                      d6
        clr.1
                      d7
@do_y
        move.1
                      LS.U_ix(a6),d0
                                                : LOAD U_ixB
        add.1
                      a1,d0
                                                   P+U_ixB
        move.1
                      d0, LS. U_ex(a6)
                                                ; SAVE U_exB
%_ob6
        GETUV
                      a1,a2,d0.d4
                                                : d4=00UV00UV (1G)
        GETY
                      (a0),a4,d4,d1,d2
                                                ; calc d2,d1 pixel
                      d2, (a3)+
        move.1
                     d1, (a3)
        move.1
        add.l
                     d5,a3
        SWAD
                      d1
        move.1
                      dl. (a3)
```

```
swap
              d2, -(a3)
d5, a3
move.1
acc. 1
              LS.Y1(a6),d0
move.1
                                         : load Yrow
              (a0,d0.w),a4,d4,d1,d2
GETY
                                        ; calc d2.d1 pixels
move.l
              d2.(a3)+
move.l
              d1.(a3)
add.l
              d5,a3
swap
             dl
move.l
              d1.(a3)
5wap
move.1
              d2, - (a3)
Swap
              d4
                                         ; next UV
addq.1
              #4,a0
                                         : next 7s
add.l
              #12,a3
             LS.Y1(a6),d0
move.1
                                        : load Yrow
GETY
              (40,d0.w),a4,d4,d1,d2
                                       ; calc d2,d1 pixels
move.1
              d1, (a3)
move.l
             d2, -(a3)
sub.1
             d5,a3
swap
              đ2
move.l
             d2, (a3) +
swap
             di
move.1
             d1, (a3)
sub.1
             d5, a3
GETY
              (a0)+,a4,d4,d1,d2
move.1
             d1, (a3)
move.l
             d2, -(a3)
SWAD
             d2
sub. 1
             d5, a3
move.l
             d2, (a3)+
swap
             dī
move.1
             d1,(a3)+
cmpa.1
             LS.7_ex(a6), a1
blt.w
             @do_x
add.l
             LS.Y_y(a6), a0
add.l
             LS.P_y(a6),a3
cmpa.1
             LS.U_ey(a6),a1
blt.w
             edo_y
movem.l
             (a7)+,d4-d7/a3-a5
                                       ; restore registers
unlk
             аб
                                        ; remove locals
rts
             ; return
ENDFUNC
macro
Y16
             LAY, LIND, LUV
move.l
             EAY. d2
                                       ; (2+) Y=Y0Y1
; (4) Y=Y0XXY1XX
lsl.l
             #5.d2
             #SPC00FC00, d2
andi.l
OI.W
             AUV. d2
                                       ; (2) Y=Y1UV
move.l
             (&IND, d2.w=4), d1
                                       ; (2+) d1=Y1
swap
             đ2
                                       : (4) Y=Y0XX
: (2) Y=Y0TV
             £07, d2
25.9
```

- 752 -

```
(&IND.d2.w*4).d2
         move.1
                                                  : (2+) d2=Y0
         move.w
                       d1.d2
                                                  ; (2) d2=Y0Y1
         endra
CUT16
         FUNC
                  EXPORT
25
         RECORD
                       8
table
         DS. L
pixmap
         DS.L
         DS.L
υ
         DS.L
         DS.L
width
         DS.L
height
        DS.L
rowByte DS.L
pixmap2 DS.L
         ENDR
LS
         RECORD
                      0, DECR
Yl
         DS.L
                                    : sizeof(short) 'Yrow
                                                                        = 2"width
U_ex
         DS.L
                      1
                                    ; x end address
                                                                        = U+U_ix
U_ey
         DS.L
                                    : y end address
                                                                        = U+width*height>>
U_ix
         DS.L
                                    ; sizeof(short)*UVrcw
                                                                        = width
۲_۲
         DS.L
                                    ; sizeof(short) 'Yrow
                                                                        = 2*width
         DS.L
                                    : 2*rowBytes-sizeof(long)*Prow = 2*rowBytes-width
LSize
         EQU
         ENDR
         a0 - Y, a1 - U, a2 - V, a3 - pixmap, a4 - table, a5 - pixmap2
d0 - rgb00, d1 - rgb01, d2 - rgb10, d3 - rgb11, d4 - spare, d6 - old0, d7
         link
                      a6, #LS.LSize
                                                 ; inc, width, fend and rowend are loca
         movem.1
                      d4-d7/a3-a5,-(a7)
                                                 ; store registers
         move.1
                      PS.Y(a6), a0
                                                 ; Y=Yc
                      PS.U(a6),a1
         move.l
                                                 ; U=Uc
         move.l
                      PS. V(a6), a2
                                                 ; V=Vc
        move. 1
                      PS.pixmap(a6),a3
                                                 ; pm=pixmap
        move.1
                      PS.table(a6),a4
                                                 ; tab=table
        adda.1
                      $$00020000.a4
                                                 : tab+=32768 (longs)
        move.1
                      PS.pixmep2(a6),a5
                                                 : pm2=pixmap2
        move.l
                      PS. width(a6), d0
                                                 : LOAD width
                                                 ; SAVE U_1X
; LOAD height
        move.1
                      d0. LS. U_ix(a6)
        move.1
                      PS.height(a6),dl
                      d0.d1
        mulu.w
                                                    width*height
        lsr.l
                      #1,d1
                                                    width*height/2
                      al,d1
d1,LS.U_ey(a6)
        add.l
                                                    U+width*height/2
        move.1
                                                 ; SAVE U_ey
        add.1
                      40,40
                                                    width*2
                      d0, LS. Y1 (a6)
        move.1
                                                  SAVE Y1
        move.l
                      d0, LS. Y_Y (a6)
                                                 ; SAVE Y_Y
        move.l
                      PS.rowByte(a6),d1
                                                  LOAD rowBytes
        add.1
                      al.al
                                                   rowBytes*2
        sub.1
                      d0,d1
                                                    rowBytes*2-width*2
        move.1
                      d1, LS. P_y (a6)
                                               ; SAVE P_Y
        move.l
                      PS.rowByte(a6),d5
                                                ; load rowBytes
        clr.l
        clr.1
                      d7
@do_y
        move. 1
                     LS.U_ix(a6),d0
                                                ; LOAD U_ixB
```

ENDFUNC

END

- 753 -

```
Engineering:KlicsCode:CompPict:Colour.a
                                                                               Page 22
        add.l
                     æ.d0
                                              : P+U_ixB
        move.1
                     d0. LS. U_ex(a6)
                                              ; SAVE U_exB
3dc_x
       GETUV
                     al.a2,d0,d4
                                              ; d4=00UV00UV (10)
                     (a0),a4,d4,d1,d2
        GETY
                                              : calc d2.d1 pixel
        move.w
                     d1,d2
        move.1
                     d2, (a3)
                     d5, a3
        add.l
        move.l
                    LS.Y1(a6).d0
                                              : load Yrow
        GETY
                     (a0,d0.w),a4,d4,d1,d2
                                             ; calc d2.d1 pixels
        move.w
                     d1,d2
        move.1
                    d2, (a3)+
        swap
                     d4
                                              ; next UV
        addq.1
                                              ; next Ys
                     #4.a0
        move.1
                     LS.Y1(a6),d0
                                              ; load Yrow
        GETY
                     (a0,d0.w),a4,d4,d1,d2
                                            ; calc d2,d1 pixels
       move.w
                    d1, d2
        move.1
                     d2, (a3)
        sub.1
                    d5, a3
        GETY
                     (a0)+,a4,d4,d1,d2
        move.w
                    d1,d2
        move.l
                    d2, (a3) +
                    LS.U_ex(a6), a1
        cmpa.1
       blt.w
                    edo_x
        add.1
                    LS.Y_y(a6),a0
LS.P_y(a6),a3
        add.l
        cmpa.1
                    LS.U_ey(a6),a1
       blt.w
                    6qo_A
       movem. 1
                    (a7)+,d4-d7/a3-a5
                                             ; restore registers
        unlk
                    a6
                                             ; remove locals
       rts
                                             ; return
```

CHECTITITE CHEET INH E OF

Engineering: KlicsCode:CompPict:Color2.a

```
© Copyright 1993 KLICS Limited
All rights reserved.
Written by: Adrian Lewis
68000 Fast RGB/YUV code
    include 'Traps.a'
    machine mc68030
    macro
    RGB2Y
             &Apixel.&AY
    d0 - pixel/r, d1 - g/2g+r, d2 - b, d3 - Y
    move.l &Apixel,d0 : pixel=*Apixel eor.l #$00808080,d0 : signed pixels
    move.b d0,d2
                               : b=pixel[]]
     ext.w
                               ; b is 8(16) bit
                               ; g=pixel[2]
; 2g is 9(16) bit
             d0,d1
    move.w
    asr.w
             #7.d1
             d0
    SWAD
                               ; r=pixel(1)
    ext.w
             d0
                               ; r is 8(16) bit
    move.w
             42,43
                               ; Y=b
    lsl.w
             43.d3
                               ; Y<<=3
    sub.w
             d2,d3
                               ; Y-=b
                               ; 2g+=r -
    add.w
             d0.d1
    add.w
             d1.d3
                               ; Y+=2g+r
    add.w
             d1.d3
                               ; Y+=2g+r
             d1, d3
    add.w
                               ; Y+=2g+r
             44.d3
    asr.w
                               ; Y>>=4
             d1,d3
    add.w
                               ; Y+=2g+r
    move.w d3.&AY
                              : AY=Y is 10(16) bit
    RGB2UV £AU, £AV
    d0 - r, d2 - b, d3 - Y, d1 - U/V
    add.w
             d0,d0
                            ; r is 9(16) bit
; b is 9(16) bit
; Y is 9(16) bit
             d2.d2
    add.w
             ‡1.d3
    asr.w
    move.w d2,d1
                               ; U=b
    sub.v
             d3, d1
                              ; U=b-Y
    move.w dl, &AU
                              ; AU=U
    move.w d0,d1
    sub.w d3,d1
move.w d1,&AV
                              ; V=r-Y
                              ; AV=V
    endm
```

Engineering:KlicsCode:CompPict:Color2.a

```
if &TYPE('seg')≠'UNDEFINED' then seg &seg endif
```

```
RGB2YUV2
            FUNC
                     EXPORT
        link
                     a6.#0
                                               ; no local variables
        movem.1
                     d4-d7/a3,-(a7)
                                               ; store registers
        move.l
                     $0008(a6).a3
                                               ; pm=pixmap
                     $000C(a6),a0
        move.1
                                                Y=Yc
        move.1
                     $0010(a6),a1
                                               ; U=Uc
        move.1
                     $0014(a6),a2
                                               ; V=Vc
        move.1
                     $0018(a6),d7
                                               ; fend=area
        asl.l
                     #2,d7
                                               ; fend<<=2
        add.1
                     a3,d7
                                               ; fend+=pm
        move.1
                     $001C(a6),d4
                                               ; width_b=width
        as1.1
                     #2.d4
                                                width_b<<=2
        move.1
                     $0020(a6),d5
                                              ; inc_b=cols
        asl.l
                     #2.d5
                                              : cols<<=2
        sub. 1
                     d4,d5
                                               ; inc_b-=width_b
edo1
        move.1
                     a3,d6
                                              ; rowend=pm
        add.l
                     d4,d6
                                              : rowend+=width_b
@do2
        rgb2y
                     (a3)+,(a0)+
                                              : rgb2y(pm++,Y++)
        rgb2uv
                     (a1)+,(a2)+
                                              ; rgb2uv(U++,V++)
        rgb2y
                     (a3)+,(a0)+
                                              ; rgb2y(pm++,Y++)
        спра. 1
                     d6.a3
                                              : rowend>pm
        blt.s
                     edo2
                                                while
        adda.l
                     45, a3
                                              ; pm+=inc_b
        move. 1
                     a3,d6
                                              : rowend=pm
        add.1
                     d4,d6
                                              ; rowend+=width_b
@do3
        rgb2y
                     (a3)+,(a0)+
                                              ; rgb2y(pm++,Y++)
        cmpa.1
                     d6.a3
                                                rowend>pm
        blt.s
                     €do3
                                                while
        adda.1
                     d5,a3
                                              : pm+=inc_b
                                                fend>pm
        cmpa.1
                     d7,a3
        blt.w
                     @do1
                                              ; while
        movem.1
                     (a7)+,d4-d7/a3
                                              ; restore registers
        unlk
                     aG
                                              ; remove locals.
        rts
                                              ; return
        ENDFUNC
       macro
       FETCHY
                    EAY, EY, ER, EG, EB
                    &AY. &Y
       move.l
                                              ; Y=*AY++
       add.l
                    LY.LR
                                             ; RR+=Y12
       add.1
                    &Y,&G
                                             ; GG+=Y12
        add.1
                    LY, LB
                                             : BB+=Y12
        endn
       MACTO
       FIXOV
                    &V, &SP1, &SP2
       move.w
                    &V, &SP1
       clr.b
                    &SP1
       andi.w
                    #$3PPP, &SP1
       SDE
                    4SP1
       btst
                   . #13,&SP1
       seq
                    &SP2
```

Engineering: KlicsCode: CompPict: Color2.a

```
&SP1.&V
        or.b
        and. w
                     SP2. LV
                     ٤V
        swap
                     &V.&SP1
        move, w
        clr.b
                     &SP1
        and1. w
                     #$3FFF,&SP1
        sne
                     ≟SP1
        btst
                     *13.43P1
        Dec
                     ≨S₽2
        or.b
                     &SP1. &V
                     &SP2, SV
        and.w
        SWAD
        endn
        macro
        OVERFLOW
                     &A. &B. &SP1. &SP2
        move.l
                     #SFF00FF00, &SP1
                                               ; spl=mask
        move.1
                     &A, &SP2
                                               : sp2=ovov (A)
        and.1
                     &SP1, &SP2
                                               : sp2=0000 (A)
        lsr.l
                     #8.4SP2
                                              ; sp2=0000 (A)
        and.l
                     &B. &SP1
                                               ; spl=0000 (B)
        or.l
                     &SP2,&SP1
                                               : spl=0000 (BABA)
        move.l
                     &A, &SP1
        or.l
                     &B. &SP1
        andi.l
                     #SFF00FF00.&SP1
        beq.s
                     Gok
                                              ; if no overflow
                    &SP2
        clr.w
                                              ; AND=0
                     &A, &SP1, &SP2
        FIXOV
                                              : Al overflow
        FIXOV
                     &B.&SP1.&SP2
                                              ; Bl overflow
eok
        endm
        macro
        MKRGB
                    ER, EG, EB, EARGB
        lsl.l
                    #8,&G
                                              : G=G0G0 (12)
        or.l
                    &B. &G
                                              ; G=GBGB (12)
        move.l
                    &R.&B
                                              ; B=0R0R (12)
        SWAD
                    &B
                                              : B=OROR (21)
        move.w
                    &G. &B
                                              ; B=0RGB (2)
        swap
                    ٤C
                                              ; G=GBGB (21)
        move.w
                    &G, &R
                                              : R=0RGB (1)
        move.l
                    &R, &ARGB
                                              : *RGB++=rgb (1)
                                              : *RGB++=rgb (2)
        move.:
                    &B, &ARGB
        endm
        macro
        DUPVAL
                    £V0, £V1
                    &V0.&V1
        move.w
                                              : v1=v0
                    EV0
        SWAD
                    £V1,£V0
        move.w
                                              ; dup v0
        move.l
                    £V0, £V1
                                              ; dup v1
        endm
        macro
        UV2RGB3
                    EAU, EAV
```

```
Engineering: KlicsCode: CompPict: Color2.a
         d1 - ra, d2 - ga, d3 - ba, d4 - rb, d5 - gb/512, d6 - bb
                      #512.d5
         move.w
                                                 : d5=512
         move.w
                      &AU,d2
                                                ; U=*AU++
         add.w
                      d2,d2
                                                 ; U is 10(16) bits
         move.w
                      d2.d3
                                                ; ba=U
         add.w
                      d3.d2
                                                ; ga=20
         add.w
                      d3.d2
                                                : ga=30
         add. w
                      d5,d3
                                                : ba+=512
         DUPVAL
                      d3.d6
                                                ; ba=bb=8B
         AST.W
                      #4,d2
                                                : ga=3U>>4
                      £AV,d1
        move.w
                                                ; V=*AV++
         add.w
                      a1.42
                                                ; ga+=V
         add.w
                      dl.dl
                                                : ra*=2
         add.w
                      d5, d1
                                                ; ra+=512
         DUPVAL
                      d1,d4
                                                ; ra=rb=RR
         sub.w
                      d2,d5
                                                ; gb=512-ga
         DUPVAL
                      d5.d2
                                                ; ga=gb=GG
        endn
        if &TYPE('seg') #'UNDEFINED' then
        seg
                      &seg
         endif
YUV2RGB2
             FUNC
                      EXPORT
25
        RECORD
pixmap
Y
        DS.L
        DS.L
U
        DS.L
                      1
٧
        DS.L
                      1
area
        DS.L
                      1
width
        DS.L
                      1
cols
        DS. L
        ENDR
LS
        RECORD
                      0, DECR
ınc
        DS.L
width
        DS.L
fend
        DS.L
                     1
count
        DS.L
                     1
LSize
        EQU
        ENDR
        a0 - Y0, a1 - Y1, a2 - U, a3 - V, a4 - pm0, a5 - pm1 d0..6 - used, d7 - count
        link
                     a6, #LS.LSize
                                               ; inc, width, fend and rowend are loca
                     d4-d7/a3-a5,-(a7)
        movem.1
                                               ; Store registers
        move.1
                     PS.pixmap(a6),a4
                                               ; pm0=pixmap
                     a4,a5
PS.Y(a6),a0
        move.1
                                               ; pml=pm0
; Y0=Yc
        move.1
                     a0,a1
PS.U(a6),a2
        move.1
                                               ; Y1=Y0
        move.1
                                               ; U=Uc
        move.1
                     PS.V(a6),a3
                                               ; V=Vc
        move.1
                     PS.area(a6),d7
                                               ; fend=area
                                               ; fend<<=2
        lsl.l
                     #2.d7
        add.l
                     a4,d7
                                               ; fend+=pm0
        move.1
                     d7.LS.fend(a6)
                                               ; save fend
```

PS.width(a6),d5

d5.d7

move.1

move.1

; width=width

; count=width

bra

```
Engineering:KlicsCode:CompPict:Color2.a
          asr.l
                        41.d7
                                                  : count>>=1
          subq.1
                        #1.d7
                                                  : count-=1
          move.1
                       d7, PS. width(a6)
                                                  : save width
          add.1
                       d5.d5
                                                  : width==2
          add.l
                       d5.al
                                                  ; Y1+=width
          add.l
                       d5,d5
                                                  : width = 2
          move.1
                       d5.LS.width(a6)
                                                  : save width
          move.1
                       PS.cols(a6),d4
                                                    inc=cols
          lsl.l
                       #2,d4
                                                  : inc<<=2
          add.1
                       d4, a5
                                                  : pml+=inc
          add.1
                       d4.d4
                                                  ; cols*=2
          sub. 1
                       d5.d4
                                                 ; inc now 2°cols-width bytes
          move. 1
                       d4, LS. inc (a6)
                                                  ; save inc
 edo
          UV2RGB3
                       (a2)+, (a3)+
                                                 ; uv2rgb(*U++,*V++)
          FETCHY
                       (a0)+,d0,d1,d2,d3
                                                 ; add Ya to RGB values ; add Yb to RGB values
          FETCHY
                       (a1)+.dC.d4.d5.d6
          move.w
                       #$3FFF.d0
                                                 ; d0=mask
          lsr.1
                       #2,d1
                                                 ; dl 8(16) bits
          and.w
                       d0.d1
                                                 ; dl masked
          lsr.1
                       #2.d2
                                                 ; d2 8(16) bits
          and.w
                       d0, d2
                                                 ; d2 masked
          1sr.1
                       #2,d3
                                                 ; d3 8(16) bits
          and.w
                       40.43
                                                 ; d3 masked
          lsr.1
                       #2,d4
                                                 ; d4 8(16) bits
          and.w
                       d0.d4
                                                 : d4 masked
          lsr.1
                       #2,d5
                                                 : d5 8(16) bics
         and, w
                       d0.d5
                                                 : d5 masked
         lsr.1
                       42.d6
                                                 ; d6 8(16) bits
         and.w
                      d0, d6
                                                 : d6 masked
         move.1
                      d1.d0
         or.l
                      d2, d0
         or.l
                      43,40
         or.l
                      d4.d0
         or.1
                      d5.d0
         or.1
                      d6, d0
         andi.1
                      #SFFUOPFOO. do
         bne.s
                      Gover
                                                ; if overflow
Ook
                      d1.d2,d3,(a4)+
         MERCA
                                                ; save RGBa
         MKRGB
                      d4, d5, d6, (a5)+
                                                ; save RGBb
         dbf
                      d7, @do
                                                : while
                      LS.inc(a6),a4
         adda.l
                                                ; pmO+=inc
                      LS.inc(26).a5
         adda.1
                                                ; pml+=inc
; Y0+=width
         adda. 1
                      LS.width(a6),a0
         exg.1
                      a0, a1
                                                ; Y1<->Y0
         move.1
                      PS.width(a6),d7
                                                ; count=width
         стра.1
                      LS.fend(a6),a4
                                                 pm0<fend
        blt.w
                      900
                                               .; while
        movem.1
                      (a7) + d4 - d7/a3 - a5
                                               ; restore registers
        unlk
                     a6
                                               ; remove locals
        rts
                                               ; return
Gover
        move.1
                     d7, LS. count (a6)
                                               ; save count
        clr.w
                                               ; AND=0
        FIXOV
                     d1,d0,d7
                                               : A overflow
                     d2.d0,d7
        FIXOV
                                               : B overflow
        FIXOV
                     43.00.07
                                               ; A overflow
        FIXOV
                     64.d0.d7
                                               ; B overflow
        FIXOV
                     d5.d0.d7
                                               ; A overflow
        FIXOV
                     d6, d0, d7
                                               ; B overflow
        move.1
                     LS. count (a6) .d7
                                               ; restore count
```

seg

&seg

Engineering:KlicsCode:CompPict:Color2.a

```
ENDFUNC
         if &TYPE('seg') = 'UNDEFINED' then
        500
                     &seg
        endif
GREY2Y FUNC
                 EXPORT
PS
        RECORD
                      8
pixmap
        DS.L
                     1
        DS.L
area
        DS.L
                      1
width
        DS.L
                     1
cols
        DS.L
                     1
        ENDR
    d0 - vvvv, d1 - v0v1, d2 - v2v3, d3 - xor, d4 - width, d5 - inc. d6 - rowend,
    a0 - pm, a1 - Y
        link
                     a6,#0
                                               ; no local variables
        movem.l
                     d4-d7,-(a7)
                                               ; store registers
        move.1
                     PS.pixmap(a6),a0
                                               ; pm=pixmap
        move.1
                     PS.Y(a6),a1
                                               ; Y=YC
        move.1
                     PS.area(a6),d7
                                               : fend=area
        add.l
                     a0,d7
                                               ; fend+=pm
        move.1
                     PS.width(a6),d4
                                               ; width_b=width
        move.1
                     PS.cols(a6),d5
                                                : inc_b=cols
                     d4,d5
#$7F7F7F7F,d3
        sub. l
                                               ; inc_b-=width_b
        move.1
                                               ; xor=$7F7F7F7F
@dol
        move.1
                     a0.d6
                                               ; rowend=pm
        add.l
                     d4.d6
                                               ; rowend+=width_b
∂do2
        move.1
                     (a0)+,d0
                                               ; vvvv=*pm
                                               : vvvv is signed : d2=v2v3
        eor.1
                     ۵۵،۵0
        move.w
                     d0,d2
        asr.w
                     #6,d2
                                               ; d2=v2 (10 bits)
; d2=v2??-
        swap
                     d2
        move.b
                     d0.d2
                                               ; d2=v2v3
        ext.v
                    ·d2
                                                 v3 extended
        Isl.w
                     #2,d2
                                                 d2=v2v3 (10 birs)
        SVAD
                     ďΰ
                                                 d0=v0v1
                     a0,a1
        move.w
                                                 dl=v0vl
                                               ; dl=v0 (10 bics); dl=v0??
        asr.w
                     #6.dl
        SWAD
                     dı
        move.b
                     d0,d1
                                               : dl=v0v1
        w. Jx9
                     dı
                                               ; v1 extended
        lsl.w
                     #2.d1
                                               ; d1=v0v1 (10 bits)
        move.l
                    d1,(al)+
                                                *Y=dl
                    d2, (a1)+
       move.1
                                               : "Y=d2
       стра.1
                    d6, a0
                                               ; rowend>pm
                    0do2
       blt.s
                                                while
       adda.1
                    d5, a0
                                              : pm+=inc_b
       cmpa.1
                    d7, a0
                                              ; fend>pm
       blt.s
                    9do1
                                              ; while
       movem.1
                     (a7)+,d4-d7
                                              ; restore registers.
       unlk
                                              ; remove locals
       rts
                                              ; return
       ENDFUNC
       if &TYPE('seg') #'UNDEFINED' then
```

פווחפדודו ודר פוורבד יחווו ל מת

endi:

- 760 Engineering:KlicsCode:CompPict:Color2.a

```
EXPORT
Y2GREY FUNC
PS
        RECORD
pixmap
        DS.L
        DS.L
                     1
height
        DS.L
width
        DS.L
cols
        DS.L
                     1
        ENDR
    d0- spare, d1 - v43, d2 - v21, d3 - spare, d4 - width, d5 - inc, d6 - count, d
    a0 - pm, al - Y
                     a6,40
        link
                                               ; no local variables
        movem. 1
                     d4-d7, -(a7)
                                               ; store registers
        move.1
                     PS.pixmap(a6),a0
                                               ; pm=pixmap
                     PS.Y(a6),a1
        move.1
                                               : Y=YC
        move.1
                     PS.height(a6),d7
                                               : long height
                     #1.d7
        subq.1
                                               ; height -= 1
                                               ; long width : long inc=cols
        move.l
                     PS.width(a6;,d4
        move.i
                     PS.cols(a6),d5
        sub.1
                     d4.d5
                                               : inc-swidth
        isr.i
                     ■2.d4
                                               ; width>>=2 (read 4 values)
        subq.l
                     *1.d4
                                               : width-=1
9dol
        move.l
                     d4,d6'
                                              : count=width
@dc2
        move.1
                     (a1) + .d0
                                              ; d0=x4x3
        move.l
                     (al)+,dl
                                              : d1=x2x1
        move.1
                     #$01FF01FF,d2
                                              : d2=511
        move.1
                     d2.d3
                                              ; d3=511
        sub. 1
                     d0,d2
                                              ; unsigned d2
        sub. 1
                     d1,d3
                                              ; unsigned d3
        lsr.l
                     #2,d2
        lsr.1
                     #2,d3
                     d2, d0
        move.1 -
        or.1
                     d3.d0
        andi.l
                     #$3F003F00.d0
        bne.s
                     gover
                                              ; if no overflow
@ok
        151.w
                     *8,d3
                                              : d3=0210
        lsl.w
                     #8,d2
                                              ; d2=0430
        isr.l
                     *8.d3
                                              ; d3=0021
                     #8.d2
        isl.1
                                              ; d2=4300
        or.i
                     d3.d2
                                              : d2=4321
        move. 1
                     d2, (a0)+
                                              : *pm=d2
        dbf
                     d6, 8do2
                                              ; while -i!=--count
        adda.l
                    d5, a0
                                              ; pm+=inc_b
        dbf
                     d7. @dol
                                              ; while -1!=--height
                     (a7)+,d4-d7
        movem.1
                                              : restore registers
        unlk
                     a6
                                              : remove locals
        rts
                                              ; return
Cover
        clr.w
                     dı
                                              ; AND=0
        FIXOV
                    d2, d0. d1
                                              ; A overflow
        FIXOV
                     d3.d0.dl
                                               B overflow
        bra.s
                     Ook
        ENDFUNC
                   -----
       macro
       GGG
                    &V, &SP1, &SP2, &AV
```

```
Engineering:KlicsCode:CompPict:Color2.a
        move.l
                      4V.4522
                                                 ; SP2=0102
        181.1
                      •8. LSP2
                                                 : SP2=1020
        cr.1
                      4V.4322
                                                 : SP2=1122
        move.1
                      4V. 4571
                                                 : SP1=0102
                      &SP1
                                                 : SP1=C201
        SWAD
                      4SP2.4SP1
        move.w
                                                 : SP1=0222
        swap
                      &SP2
                                                 ; SP2=2211
        move.w
                      4SP2.4V
                                                 ; V=0111
        move.1
                      &V. &AV
                                                 ; *pm=V
                      ESP1, EAV
                                                 : *pm=SP1
        move.1
        endm
        if &TYPE('seg') = 'UNDEFINED' then
        seg
                      £seg
        endif
Y2GGG
        FUNC
                 EXPORT
PS
        RECORD
pixmap
        DS.L
        DS.L
                      1
lines
        DS.L
                      1
width
        DS.L
                      1
cols
         DS.L
                      1
        ENDR
    d0 - v, d4 - width, d5 - inc, d6 - count, d7 - lines a0 - pm, a1 - \Upsilon
                      a6.#0
                                                 ; no local variables
        link
                      d4-d7,-(a7)
        movem.1
                                                 ; store registers
        move.1
                      PS.pixmap(a6),a0
                                                 ; pm=pixmap
        move.1
                      PS.Y(a6),a1
                                                 ; Y=YC
                                                 ; long lines
         move.1
                      PS.lines(a6),d7
         subq.1
                      #1,d7
                                                   lines-=1
        move.1
                      PS.width(a6).d4
                                                 : long width
        move.1
                      PS.cols(a6),d5
                                                 ; inc=cols
                      d4.d5
                                                 ; inc-=width
        sub.1
                                                 ; inc (bytes)
         151.1
                      #2.d5
                                                 ; width>>=2
                      42.d4
         lsr.l
                                                 ; width-=1
                      41,d4
         subq.1
                                                 ; count=width
4do1
                      d4.d6
        move. 1
                                                 : d0=x1x2 (10 bits signed)
: d1=x3x4 (10 bits)
                      (a1)+,d0
(a1)+,d1
+$02000200,d3
3do2
        move.l
        move.1
                                                : d3=plus
: d0=x1x2 (unsigned)
        move.l
        add.l
                      0b, Eb
                                                 ; dl=x3x4 (unsigned)
         add.l
                      d3.d1
                                                 ; d0=x1x2 (10.8 bits)
         lsr.l
                      #2,d0
         lsr.l
                      #2.dl
                                                 ; dl=x3x4 (10,8 bits)
                      #$3FFF,d2
                                                 ; d2=mask
        move.w
                      d2, d0
                                                 ; mask d0
         and.w
                      d2,d1
d0,d2
                                                 ; mask dl
         and. v
        move.1
                      d1,d2
         or.l
         andi.l
                      *SFF00FF00,d2
                                                 : if no overflow
                      20ver
         bne.s
                      d0,d2,d3,(a0)+
Cok
         GGG
         GGG
                      d1.d2.d3, (a0)+
                                                 ; while -1!=--count
         dbf
                      d6,9do2
                                                 ; pm+=inc_b
; while -1!=--lines
         adda.l
                      d5.a0
         ಚಿದ್ದ
                      d7.9d:1
```

area

□s.l

ı

```
Engineering:KlicsCode:CompPict:Color2.a
         movem.l
                      (a7) + . d4 - d7
                                                : rescore registers
         unlk
                      a 6
                                                 : remove locals
         rts
                                                : return
         cir.w
 iover
                      43
                                                C=CNA :
         FIXOV
                      d0.d2.d3
                                                . A overflow
         FIXOV
                      d1.d2.d3
                                                : B overflow
         bra.w
                      ĕок
         ENDFUNC
         macro
         MXRGB2
                      ER, EG, EB, EARGB, EROW, EXX
         1s1.1
                      48, &G
                                                : G=G0G0 (12)
         or.l
                      4B, 4G
                                                : G=GBGB (12)
         move.1
                      SR. SB
                                                ; B=OROR (12)
         SWAD
                      &B
                                                ; B=0R0R (21)
         move.w
                      4G, 4B
                                                : B=0RGB (2)
         swap
                      £G
                                                : G=GBGB (21)
         move.w
                      EG, ER
                                                ; R=0RGB (1)
         andi.l
                      *SFFFEFEFE, &R
                                                : 7 bits for interpolation
         and: .1
                      *SFFFEFEFE, &B
                                                ; 7 bits for interpolation
         move.1
                      &R. €G
                                                ; G=RGB(1)
         add.1
                                               ; G+=RGB(2)
; G/=2
                      &B, &G
         lsr.1
                      *1,4G
         move.l
                      &B. &XX
                                                : XX=RGB(2)
         sub. l
                      ER, EXX
                                                ; XX-=RGB(1)
         lsr.1
                      41, &XX
                                                ; XX/=2
         add.1
                      43.4XX
                                               : XX+=B
         move.1
                      ER, (LARGE)+
                                               ; *RGB++=rgb (1)
; *RGB++=rgb (1.5)
                      &G. (&ARGB)+
         move.1 -
         move.1
                      &B. (&ARGB) +
                                               : *RGB++=rgb (2)
        move.1
                      &B, (&ARGB) +
                                               ; *RGB++=rgb (2.5)
         add.1
                     &ROW, &ARGB
         sub. 1
                      $16, &ARGB
        move.1
                     &R, (&ARGB)+
                                               : *RGB++=rgb (1)
        move.l
                     &G. (&ARGB)+
                                               ; *RGB++*rgb (1.5)
        move.1
                     &B, (&ARGB) -
                                               : *RGB++=rgb (2)
        move.1
                     EB, (LARCE)+
                                               ; *RGB++=rgb (2.5)
        sub. i
                     SROW, SARGE
        endm
        if LTYPE('seg') #'UNDEFINED' then
        sea
                     £seg
        endif
YUV2RGB3
            FUNC
                    - EXPORT
PS
        RECORD
                     8
pixmap
        DS.L
Y
        DS.L
        DS.L
v
        DS.L
                     1
```

Engineering: KlicsCode: CompPict: Color1.a

```
width
        DS.L
        53.L
cols
        ENDR
LS
        RECORD
                      0. DECR
inc
        DS.L
        DS.L
width
tend
        DS.L
                      1
count
        DS.L
row
        DS.L
                      1
Lâize
        EOU
        ENDR
        a0 - Y0, a1 - Y1, a2 - U, a3 - V, a4 - pm0, a5 - pm1 d0. 6 - used, d7 - count
                                                ; inc. width, fend and rowend are loca
                      a6.#LS.LSize
        link
                      d4-d7/a3-a5,-(a7)
                                                ; store registers
        movem. 1
                                                ; pm0=pixmap
        move.1
                      PS.pixmap(a6),a4
        move.1
                      a4.a5
                                                ; pml=pm0
                                                ; Y0=YC
        move.1
                      PS.Y(a6), a0
        move.1
                      a0.al
                                                ; Y1=Y0
                                                ; U=Uc
                      PS.U(a6),a2
        move. 1
                      PS. V(a6), a3
                                                ; V=Vc
        move.1
                                                ; fend=area
                      PS.area(a6),d7
        move.1
                                                ; fend<<=2
         151.1
                      #2.d7
         add.l
                      44.d7
                                                 ; fend+=pm0
        move.1
                      d7, LS. fend(a6)
                                                ; save fend
                                                 ; width=width
        move.1
                      PS.width(a6),d5
         move.l
                      d5,d7
                                                  count=width
         asr.l
                      #1.d7
                                                 : count>>=1
         subq.1
                      *1.d7
                                                ; count-=1
         move.1
                      d7, PS. width(a6)
                                                ; save width
                      d5.d5
         add.l
                                                ; width =2
         add.l
                      d5,a1
                                                ; Yl+=width
                                                ; width==2
         add. 1
                      d5,d5
                                                ; save width
                      d5, LS. width (a6)
         move.1
         move.1
                      PS.cols(a6),d4
                                                ; inc=cols
         1s1.1
                      #2.d4
                                                ; inc<<=2
                      d4, L5. row(a6)
                                                ; "NEW save row
         move.l
         add.1
                      d4.a5
                                                ; pml+=inc
         add.1
                      d4, a5
                                                ; "NEW pml+=inc
         add.l
                      d4, d4
                                                : cols =2
                      d4.d4
                                                ; "NEW cols"=2
                                                ; inc now 4*cols-width bytes
         sub. 1
                      d5,d4
                      d5,d4
d4,LS.inc(a6)
                                                ; *NEW inc now 4*cols-width bytes (wid
         sub. 1
                                                ; save inc
         move.1
                                                ; uv2rgb(*U++,*V++)
         UV2RGB3
                      (a2)+, (a3)+
@do
                      (a0)+,d0,d1,d2,d3
                                                ; add Ya to RGB values
         FETCHY
                                                ; add Yb to RGB values
         FETCHY
                      (a1)+,d0,d4,d5,d6
                      #$3FFF,d0
                                                ; d0=mask
         move.w
                                                ; d1 8(16) bits
         lsr.l
                      #2.dl
                                                ; dl masked
         and.w
                      d0,d1
                                                ; d2 8(16) bits
                      #2.d2
         lsr.l
         and.w
                      d0, d2
                                                ; d2 masked
         lsr.l
                      #2,d3
                                                ; d3 8(16) bits
                      d0, d3
                                                ; d3 masked
         and.w
                      #2,d4
                                                ; d4 8(16) bits
         lsr.1
                      d0,d4
                                                ; d4 masked
         and.v
                      42.45
                                                  45 8(15) bits
         lsr.1
```

move. w

Engineering:KlicsCode:CompPict:Color3.3

```
; d5 masked
         and.w
                        d0.d5
                                                     : d6 8(16) Dits
                        •2.d5
         lsr.l
                                                     : d6 masked
                        d0.d5
         and.w
                        d1.d0
         πeve.1
                        d2.d0
         or.1
                        d3.d0
         cr.1
                        d4.d0
         cr.l
                        d5.d0
         or.l
                        d6.d0
         or.l
                        *SFF00FF00.d0
         andi.l
                                                      ; if overflow
                        @over
         bne.w
                                                          : NEW save RGBa
                        d1.d2.d3.a4.LS.row(a6).d0
         MKRGB2
Jok
                                                          : NEW save RGBb
                        d4.d5,d6,a5,LS.row(a6),d0
         MKRGB2
                                                      ; while
         ತಿಯ
                        d7,9do
                                                      : pm0+=inc
         adda.1
                        LS.inc(a6).a4
                                                      : pml+=inc
                        LS.inc(a6).a5
         adda. 1
                                                        Y0+=width
                        LS.width(a6),a0
         adda.1
                                                      ; Y1<->YG
                        a0.al
         exg.l
                                                      ; count=width .
                        PS.width(a6),d7
         move. ì
                                                      ; pm0<fer.d
                         LS.fend(a6),a4
         cmpa. 1
                                                      ; while
                        edo
         blr.w
                                                      ; restore registers
                         (a7)+,d4-d7/a3-a5
          movem.1
                                                      ; remove locais
          unlk
                        a6
                                                        return
          TTS
                                                      ; save count -
                         d7, LS. count (a6)
          move.1
3over
                                                      : AND=0
                         d7
          clr.w
                                                      ; A overflow
                         di. do. d7
          FIXOV
                                                      ; B overflow
          FIXOV
                         d2, d0, d7
                                                      : A overflow
          FIXOV
                         d3, d0, d7
                                                      ; B overflow
          FIXOV
                         d4.d0.d7
                                                      ; A overflow
          FIXOV
                         d5.d0.d7
                                                      ; B overflow
                         d6, d0, d7
          FIXOV
                                                      ; restore count
                         LS. count (a6), d7
          move.1
                         eok
          bra
          EMPTUNC
          macro
                         EAY, EY, ER. EG. EB
          FETCHY2
                         LAY, LY
          move.1
          asr.w
                         #2.&Y
          swap
                         ٤Y
                                                      ;Y is -128 to +127
;RED, Get (Y+ 2V + 512) for Red = (Y +
;GREEN, Get (Y + (512 - (6U/16)) - V)
;BLUE,Get (Y + (2U + 512) for Blue = (
                         #2.&Y
          asr.W
          SWAD
                         LY. LR
           add. 1
          add.1
                         &Y.&G
                         &Y, &B
           add. 1
           endm
  . . . . . . . . . . . . . . . . . . .
           macro
           UV2RGB4
                         EAU, EAV
                          EAU. d2
                                                      :BLUE.Get (2U + 512)/4 for Blue = (Y + :Dup for second pair :GREEN. Get (512 - (6U/16))/4 for Green
                          #$03FF,d2
           and.w
                          (a6.d2.w-8),d3
           move.1
           move.1
                          d3,d6
                          4(a6,d2.w=8).d5
           move.1
                          Ib.VA2
```

Engineering: KlicsCode: CompPict: Color2.a

```
d1,d4
        move.w
        asr. w
                      #2,d1
        sub.w
                      41.45
                                                 :GREEN. Get (512 - (6U/16) - V)/4 for
        move.w
                      15,d2
        swap
                      15
        move.w
                      d2.d5
                                                 ;Dup for second pair
        move.i
                      ds,d2
                      #$03FF,d4
        and.w
        move.1
                      (a6.d4.w.8).d4
                                                :RED, Get (2V + 512)/4 for Red = (Y +
        move.1
                      d4.d1
        endm
MKRGB2SUB FUNC
                      EXPORT
        MKRGB2
                      d1.d2.d3.a4.d7.d0
                                            : "NEW save RGBa
        MKRGB2
                      d4,d5,d6.a5,d7,d0
                                            : "NEW save RGBb
        rts
        ENDFUNC
OVERSUB FUNC
                      EXPORT
                      d1,d0
d2,d0
        move.l
        or.l
                      d3.d9
        or.l
        or.l
                      d4.d0
        or.l
                      d5,d0
        or.l
                      d6.d0
                      *SFF00FF00.dC
         andi.l
        bne.s
                      @over
                                                 ; if overflow
0ok
        rts
                      d7, -(sp)
fover
        move.1
                                                 ; save count
                      d7
        clr.w
                                                ; AND=0
                      d1.d0.d7
                                                ; A overflow
; B overflow
        FIXOV .
                      d2.d0,d7
        FIXOV
                      d3,d0,d7
d4,d0,d7
        FIXOV
                                                 ; A overflow
        FIXOV
                                                ; B overflow
        FIXOV
                      45.40,47
46,40,47
                                                : A overflow : B overflow
        FIXOV
        move.1
                      (sp)+,d7
                                                : restore count
        bra
                      9ok
        ENDFUNC
UV2RGB4SUB FUNC
                      EXPORT
        UV2RGB4
                      (a2) +, (a3) +
                                                ; uv2rgb(*U++,*V++)
        rts
        ENDFUNC
FETCHY2SUB FUNC
                      EXPORT
                                                ; add Ya to RGB values
; add Yb to RGB values
        FETCHY2
                      (a0)+,d0,d1,d2,d3
        FETCHY2
                      (al)+,d0,d4,d5,d6
        rts
        ENDFUNC
        if &TYPE('seg') = 'INDEFINED' then
```

Engineering: KlicsCode: CompPict: Colorl.a

```
4seq
         seg
        endit
                      EXPORT
YUVZRGB5
             FUNC
95
         RECORD
                      3
        DS.L
DS.L
Table
pixmap
Y
                      1
         DS.L
DS.L
U
         DS.L
area
         DS.L
width
         DS.L
cols
         DS.L
                      1
         ENDR
LS
         RECORD
                      0, DECR
ınc
         DS.L
                      1
width
         DS.L
iend
         DS.L
count
         DS.L
TOW
         DS.L
                      1
LSize
         EOU
         ENDR
         a0 - Y0. a1 - Y1, a2 - U, a3 - V, a4 - pm0, a5 - pm1
         d0..6 - used, d7 - count
                                                 ; inc. width, fend and rowend are loca
         link
                      a6, #LS.LSize
         movem.l
                      d4-d7/a3-a5,-(a7)
                                                 ; store registers
                      PS.pixmap(a6),a4
                                                 ; pm0=pixmap
         move.1
                                                   Dury=Durg
         move. 1
                      a4.a5
PS.Y(a6),a0
                                                   Y0=YC
         move. 1
                      a0.al
P5.U(a6),a2
                                                   Y1=Y0
         move.1
                                                 : U=UC
         move. 1
                      PS.V(a6), a3
                                                 ; V=Vc
         move.l
                                                 : fend=area
                      PS.area(a6),d7
         move.l
                                                   fend<<=2
                      #2,d7
         1s1.1
                                                 ; fend+=pm0
         add.l
                      a4,d7
                                                 ; save fend
         move.1
                      d7, LS. fend(a6)
                                                   width=width
         move.1
                      PS.width(a6).d5
         move.1
                      d5.d7
                                                   count=width
                      #1.d7
                                                   count>>=1
         asr.l
                      #1.d7
                                                 : count-=1
         subq.1
                      d7.PS.width(a6)
                                                 ; save width
         move.1
         add.l
                      d5,d5
                                                 ; width==2
                                                 ; Y1+=width
         add.l
                      d5, a1
         add.l
                      d5,d5
                                                 ; width*=2
                      d5, LS. width(a6)
                                                 ; save width
         move.1
                      PS.cols(a6),d4
#2,d4
                                                 : inc=cols
         move.1
                                                 ; inc<<=2
         1s1.1
                      d4, LS. row(a6)
                                                 ; "NEW save row
         move.l
                                                 ; pml+=inc
         add.l
                      d4.a5
                                                 : "NEW pml+=inc
: cols"=2
         add.1
                      d4.a5
         add.1
                      d4.d4
                                                 .. NEW cols.=3
                                                 : inc now 4*cols-width bytes
         add.l
                      d4,d4
         sub.1
                      d5,d4
                                                        inc now 4 cols-width bytes (wid
         sub. l
                      d5,d4
                                                 : save inc
         move.1
                      d4.LS.inc(a6) -
94:
         move.1
                      47, - (ap)
```

```
Engineering:KlicsCode:CompPict:Color2.a
                     a6, -(sp)
        move. 1
        move. I
                     LS. row(a6) .d7
                     PS.Table(a6:,a6 -
                                               ; uv2rgb(*U++,*Y++)
        UV2RGB4
                     (a2) +, (a3) +
                                               ; add Ya to RGB values
                     (a0)+,d0,d1,d2,d3
        FETCHY2
                                               ; add Yb to RGB values
                     (al)+,d0,d4,d5,d6
        FETCHY?
                     d1.d0
        move.i
                     d2.d0
        or.l
                     d3.d0
        or.1
        or.:
                     d4.d0
        or.l
                     d5.d0
                     d6.d0
        or.1
        andi.l
                     *SF700FF00, d0
                                               ; if overflow
                     gover
        bne.w
                                           ; "NEW save RGBa
                     d1.d2.d3.a4.d7.d0
@ok
        MKP.GB2
                                           ; NEW save RGBb
        MKRGB2
                     d4.d5.d6.a5.d7.d0
                     (sp)+,a6
        move.l
        move.1
                     (sp)+,d7
                                               : while
                     d7,3do
        db f
                                               ; pm0+=inc
        adda.1
                     LS.inc(a6),a4
                                               ; pml+=inc
; Y0+=width
         adda.l
                     LS.inc(a6),a5
         adda.:
                     LS.width(a6).a0
                                                ; Y1<->Y0
                     a0,al
         exg.l
                                               ; count-width
                     PS.width(a6),d7
        move.1
                                               ; pm0<fend
                     LS.fend(a6),a4
        cmpa.1
                                               ; while
                     0do
         blt.s
                                               ; restore registers
                      (a7)+,d4-d7/a3-a5
         movem. 1
                                               : remove locals
         unlk
                     a6
                                               ; return
         rts
                                               ; save count
                     d7, LS. count (a6)
@over
        move.l
                                               ; AND=0
                     d7
         clr.w
                                               ; A overflow
                     d1.d0.d7
         FIXOV
                                              : B overflow
: A overflow
         FIXOV
                     d2, d0, d7
         FIXOV
                      43,40,47
                                               ; B overflow
                      d4.d0.d7
         FIXOV
                                               ; A overflow
                      d5.d0.d7
         FIXOV
                                               ; B overflow
                      d6, d0, d7
         FIXOV
                                               ; restore count
         move.1
                      LS. count (a6) . d?
                      Bok
         cra
         ENDFUNC
```

Engineering: KlicsCode: CompFist: Clut.s

```
© Copyright 1993 KLICS Limited
   All rights reserved.
   Written by: Adrian Lewis
 .....;
    Analyse CLUT setup and pick appropriate
    YUV->RGB converter/display driver. Create
    any tables necessary.
*include <QuickDraw.h>
=include <Memory.h>
•define Y_LEVELS
=define UV_LEVELS
                      16
*define absv(v) ((v)<0?-(v):(v))
*define NewPointer(ptr,type.size) \
    saveZone=GetZone(); \
    SetZone(SystemZone()); \
    if (nil==(ptr=(type)NewPtr(size))) ( >
         SetZone(ApplicZone()): \
         if (nil==(ptr=(type)NewPtr(size))) { \
             SetZone(saveZone); \
             return(MemoryError()); \
        ) \
    SetZone(saveZone);
typedef struct (
cnar y, u, v:
) YUV_Clut:
unsigned char .
ColourClut(CTabHandle clut)
   int size, y, u, v, r, g, b, i; unsigned char table;
    YUV_Clue
                 *yuv_clut;
     size=(*clut)->ctSize;
     table=(unsigned char *)NewPtr(Y_LEVELS*UV_LEVELS);
    yuv_clut=(YUV_Clut *)NewPtr(size*sizeof(YUV_Clut));
     for(i=0;i<=size:i++) (
         r=((*clut)->ctTable(i).rgb.red>>8)-128;
         g=((*clut)->ctTable(i).rgb.green>>8)-128;
         b=((*clut)->ctTable(i].rgb.blue>>8)-128;
         yuv_clut[i].y= (306*r + 601*g + 117*b)>>10;
yuv_clut[i].u= (512*r - 429*g - 83*b)>>10;
yuv_clut[i].v= (-173*r - 339*g + 512*b)>>10;
     for(y=-Y_LEVELS/2;y<Y_LEVELS/2-1;y++)
     for(u=-UV_LEVELS/2;u<UV_LEVELS/2-1;u++)
for(v=-UV_LEVELS/2;v<UV_LEVELS/2-1;v++) (
                 index, error, error2, points. Y. U. V:
```

Engineering: KlicsCode: CompPict: Clut.c

```
Y=y<<4:
         U=u<<5;
         Y=v<<5;
         index=0:
         error=131072;
         error2=131072:
         points=0;
         for(i=0:i<=size:i++) (</pre>
              int pts=0, err=0;
              if (yuv_clut(i).y>=Y && yuv_clut(i).y<Y+16)
                  pts+=1;
              err+=absv(yuv_clut[i].y-Y);
              if (yuv_clut(i).u>=U && yuv_clut(i).u<U+32;
                  pts+=1;
              err+=absv(yuv_clut(i).u-U);
              if (yuv_clut(i).v>=V && yuv_clut(i).v<V+32)</pre>
                   pcs+=1:
              err+=absv(yuv_clut(i).v-V);
              if (pts>points || (pts==points && err<error)) {
                   error=err;
                   index=i;
                   points=pts:
              )
         i=((y&0x1F)<<8))((u&0xF)<<4))(v&0xF);
         table(i)=(unsigned char)index:
    DisposePtr((Ptr)yuv_clut);
    return table:
typedef union (
    long pixel: unsigned char
                       rgb(4);
Pixel:
unsigned long *
ColourClut (CTabHandle clut)
             size. y, u, v, r, g, b, ro, go, bo.i;
*table;
     long
     Pixel
    size=(*clut)->ctSize:
table=(Pixel *)NewPtr(Y_LEVELS*UV_LEVELS*UV_LEVELS*sizeof(long));
     for(y=-Y_LEVELS/2;y<Y_LEVELS/2-1;y++)
for(u=-UV_LEVELS/2;u<UV_LEVELS/2-1;u++)
     for(v=-UV_LEVELS/2:v<UV_LEVELS/2-1:v++) (
          Pixel
                  px;
                   base, dith;
          long
          r = 32768L + ((y<<9) + 1436L^u <<2);

g = 32768L + ((y<<9) - 731L^u - 352L^v <<2);

b = 32768L + ((y<<9) + 1815L^v <<2);
          r=r<0?0:r>65534?65534:r;
          g=g<0?0:g>65534?65534:g;
b=b<0?0:b>65534?f5534:b;
```

Engineering:KlicsCode:CompPict:Clut.c

```
ro=r$13107: r=r/13107;
         go=g%13107: g=g/13107:
         bo=b$13107: b=b/13107:
         base=215-(35*r+6*g+b);
         dith=base-(ro>2621?36:0)-(gc>7863?6:0)-(bo>10484?1:0);
         px.rgb(0)=dith==215?255:dith:
         dith=base-(ro>5242?36:0)-(go>10484?6:0)-(bo>2621?1:0);
         px.rgb[1]=dich==215?255:dich:
         dith=base-(ro>7863?36:0)-(go>2621?6:0)-/bo>5242?1:0);
         px.rgb(2)=dith==215?255:dith:
         dith=base-(ro>10484?36:0)-(go>5242?6:0)-(bo>7863?1:0);
         px.rgb[3]=dith==215?255:dith;
         i=((y60x3F)<<8))((u60xF)<<4))(v60xF);
         cable(i).pixel=px.pixel;
    return (unsigned long*)table;
typedef struct (
iong red, green, blue;
} RGBError;
OSErr ColourClut(Pixel **table)
             y, u, v, r, g, b, i;
r *err;
    RGBError
             saveZone:
    NewPointer(*table.Pixel*.Y_LEVELS*UV_LEVELS*UV_LEVELS*sizeof(long)); /* 64k ta
    NewPointer(err, RGBError*, Y_LEVELS*UV_LEVELS*UV_LEVELS*sizeof(RGBError));
    for(i=0;i<4:i++)
    for(y=-Y_LEVELS/2;y<Y_LEVELS/2;y++)
for(u=-UV_LEVELS/2;u<UV_LEVELS/2;u++)
for(y=-UV_LEVELS/2;v<UV_LEVELS/2;v++) (
        RGBColor src. dst:
        long
                 index.in:
        index=((y&0x3F)<<8)|((u&0xF)<<4)|(v&0xF);
        r = 32768L + ((y<<9) + (1436L*u) <<2);

g = 32768L + ((y<<9) - (731L*u) - (352L*v) <<2);

b = 32768L + ((y<<9) + (1815L*v) <<2);
        if (i>0) (
             r-=err(index).red;
             g-=err(index).green:
             b-=err(index).blue;
        }
        src.red=r<0?0:r>65534?65534:r;
        src.green=g<070:g>65534?65534:g;
        src.blue=b<0?0:b>65534?65534:b;
        (*table)(index).rgb[i]=(unsigned_char(Color2Index(Asrc))
```

```
→ Engineering: KlicsCode: CompPict: Clut.c
        Index2Color((*table)(index).rgb(i).4dst);
        err(index).red=dst.red-src.red:
err(index).green+dst.green-src.green;
        err(index).blue=dst.blue=src.blue;
   DisposePtr((Ptr)err);
    return(noErr):
typedef struct (
    short pel(2):
) Pix16;
typedef struct (
    unsigned char pel(4);
) 21x8:
*define YS 64
*define UVS 32
OSErr Colour8(Pix8 **table)
            y, u, v, r, g, b, i;
r *err;
     long
     RGBETTOT
             saveZone;
     THE
    NewPointer(*table.Pix8*,YS*UVS*UVS*sizeof(Pix8)); /* 128k table */
NewPointer(err,RGBError*,YS*UVS*UVS*sizeof(RGBError));
     for(i=0:i<4:i++)
     for (y=-YS/2;y<YS/2;y++)
     for (u=-UVS/2;u<UVS/2;u++)
     for(v=-UVS/2; v<UVS/2; v++) {
         RGBColor src. dst;
                  index;
         long
          index=(y<<10)|((u&0x1F)<<5)|(v&0x1F);
         if (i>0) (
              r-merr(32768+index).red:
              g-serr(32768+index).green:
              b-serr[32768+index].blue;
          src.red=r<0?0:r>65534?65534:r;
          src.green=g<0?0:g>65534?65534:g;
          src.blue=b<0?0:b>65534?65534:b;
          (*table)[32768+index].pel[i]=(unsigned char)Color2Index(&src);
Index2Color((*table)[32768+index].pel[i],&dst);
          err[32768+index].red=dst.red-src.red;
          err[32768+index].green=dst.green-src.green;
          err[32768+index].blue=dst.blue-src.blue;
      DisposePtr((Ptr)err);
      return (noErr);
```

```
Engineering: KlicsCode:CompPict:Clut.c
OSErr Colour16(Pix16 **table)
             y, u, v, t, g, b, 1;
     RGBEIror
                  *err:
     THE
             saveZone:
    NewPointer("table.Pix16",YS"UVS"UVS"sizeof(Pix16)): /* 128k table "/ NewPointer(err.RGBError",YS"UVS"UVS"sizeof(RGBError)):
     fcr(i=0:1<2:i++)
     for(y=-YS/2;y<YS/2;y++)
     fcriu=-UVS/2;u<UVS/2;u++)
     tcr(v=-UVS/2; v<UVS/2; v++) (</pre>
         RGBColor
                      src. dst:
                  index:
         long
         index=(y<<10)1((u&0x1F)<<5)1(v&0x1F);
         r = 32768L + ((y<<10) + (1436L*u) <<1);

g = 32768L + ((y<<10) - (731L*u) - (352L*v) <<1);

b = 32768L + ((y<<10) + (1915L*v) <<1);
         if (i>0) (
              r-=err(32768+index).red:
              g-=err(32768+index).green;
              b-=err(32768+index;.blue:
         src.red=r<0?0:r>65534?65534:r;
src.green=g<0?0:g>65534?65534:g;
         src.blue=b<0?0:b>65534?65534:b:
         dst.red= src.red&0xF900;
          dst.green= src.green&CxF800:
         dst.blue= src.blue&0xF800;
         (*table)[32766+index].pel[i]=(dst.red>>1)|(dst.green>>6)|(dst.blue>>11);
         err(32768+index).red=dst.red-src.red:
         err(32768+index).green=dst.green-src.green;
         err(32768+index).blue=dst.blue-src.blue;
     DisposePtr((Ptr)err):
     return(noErr);
 Bcolean
 GreyClut(CTabHandle clut)
     Boolean result=true;
     int
             i, size;
     size=("clut)->ctSize:
     for(i=0;i<=size && result;i++) {</pre>
                  r,g,b;
          int
          r=(*clut)->ctTable(i).rgb.red;
          g=(*clut)->ctTable(i).rgb.green;
          b=(*clut)->ctTable(i).rgb.blue;
          result=(r==g && g==b);
```

Engineering:KlicsCode:CompPict:Clut.c

return result;

Engineering:KlicsCode:CompPict:Bits3.h

```
.........
   O Copyright 1993 KLICS Limited
   All rights reserved.
   Written by: Adrian Lewis
Bits3.h: fast bit read/write definitions
               define static variables
   buf_use
              initialise vars for write
   buf_winit
              initialise wars for read
   buf_rinic
   buf_set
               set current bit
   buf_get
               get current bit
               increment write buffer
   buf_winc
               increment read buffer
   buf_rine
   buf_size
buf_flush
               fullness of buffer in bytes
              flush buffer
   User defined macro/function buf_over must be defined in case of buffer overflo
typedef struct (
                  *buf:
   unsigned long
   morou (
       unsigned long mask;
       long
               puo:
    ) index;
    unsigned long
                 *ptr, data, size;
) Buffer, *Buf:
#define buf_winit(buf) \
   buf->index.mask=0x80000000; \
   buf->ptr=&buf->buf(0); \
   buf->data=0;
#define buf_rinit(buf) \
   buf->index.bno=0; \
   buf->ptr=&buf->buf(0);
*define buf_set(buf) \
   buf->data != buf->index.mask;
=define buf_get(buf) \
   0!=(buf->data & (1<<buf->index.bno) )
#define buf_winc(buf) \
    if (buf->index.mask==1) ( \
        *buf->ptr=buf->data: \
       buf->data=0: \
       buf->index.mask=0x80000000; \
       buf->ptr++: \
    } else buf->index.mask >>= 1;
#define buf_rinc(buf) \
    if (--(buf->index.bno)<0) { \
       buf->data=*buf->ptr++; \
       buf->index.bno=31; \
/* buf_size only valid after buf_flush */
```

Engineering:KlicsCode:CompPict:Bitsl.a

```
© Copyright 1993 KLICS Limited
   All rights reserved.
   Written by: Adrian Lewis
•-----
   63000 Bit buffer code (Bits2.h)
Macros:
      buf_winir &ptr.&data,&mask,&buf
buf_rinit &ptr.&bno.&buf
                 £data,£mask
      buf_set
                &data,&bno
&ptr.&data,&mask
      buf_get
      buf_winc
      buf_rinc &ptr.&data,&index buf_flush &ptr.&data,&mask
                  &ptr.&data,&index
       macro
       buf_winit &ptr.&data,&mask,&buf
                   $$80000000, £mask
                                         ; mask=100..
       move.1
                                         : ptr=buf
: data=0
                   &buf, &ptr
       move.1
                   &data
       clr.1
       endm
      -----
       macro
       buf_rinit &ptr.&bno.&buf
                   &bno
                                         ; bno=0
                                         ; ptr=buf
       move.1
                  &buf,&ptr
       endm
       macro
                  &data.&mask
       bu:_set
                                         ; data != mask
                  &mask,&data
       cr.1
       endm
       macro
       buf_get
                  &data,&bno
                   #1,&bno
       subq.b
       best
                  &bno.&data
       endm
       macro
       buf_winc
                   &ptr.&data,&mask
                                          ; mask>>=1
       lsr.l
                   #1.&mask
                                          ; if non-zero continue
                   PCONT
       bne.s
                                         ; *ptr++=data
; data=0
       move.1
                   &data,(&ptr)+
                  &data
#5900000000, Emask
       clr.1
                                         magk=100.
       movre. 1
```

Engineering:KlicsCode:CompPict:Bits3.a

3 - 51			
•	endm		
•	nacro bui_rinc	iper, idata, ibno	
•	cmpi.b bge.s swap move.w add.b	<pre>#16.&bno @cont &data (&ptr)+.&data #16.&bno</pre>	; data=*ptr++ : bno+=16
@cont			•
_	endm		
	macro buf_flush	&ptr.&data.&mask	
•	cmp.l beq.s move.l	#\$80000000,&mask 9cont &data,{&ptr}+	<pre>; mask-8000000? ; if buffer empty continue ; *ptr++*data</pre>
. •	endm	•	

...........

*define BwdE2(add:2.d&G.dAH.dBH) '

Engineering: KlicsCode: CompPict: Backward.c

```
. Ø Copyright 1993 KLICS Limited
* All rights reserved.
* Written by: Adrian Lewis
Extra fast Backward\convolver
   New wavelet coeffs : 3 5 1 1, 1 2 1, 1 1
   Optimized for speed:
       dirn - Faise
       src/dst octave == 0
*define BwdS0(addr0.dAG.dAH.dBH) \
   v=*(short *)addr0; \
   dAG= -v: \
   dAH= v; \
    dBH= v<<1; \
*define BwdS1(addr1,addr0.dAG.dAH,dEH) \
   v=*(short *)addrl; \
dBH+= v>>1; \
    dAG+= v-(vs=v<<1); \
    dAH-= v+(vs<<=1); \
    *(short *)addr0=dBH>>1;
#define Bwd2(addr2,dAG,dAH,dBG,dBH) \
    v=*(short *)addr2; \
    dBG= -v; \
dBH= v; \
    dAH+= v+(vs=v<<1); \
    dAG+= v+(v5<<=1);
#define Bwd3(addr3.addr2.addr1.dAG,dAH.dBG.dBH) \
    v=*(short *)addr3; \
    dAH+= v; \
    dAG+= v; \
    daG+= v+(vs=v<<1): \
    dBH-= v+(vs<<=1); \
    "(short ')addrl=(dAH-1)>>2: \
"(short ')addr2=(dAG-1)>>2:
#define Bwd0(addr0.dAG.dAH.dBG.dBH) \
    v=*(short *)addr0; \
    dAG= -v; \
dAH= v; \
    dBH+= v+(vs=v<<1); \
    dBG+= v+(vs<<=1);
#define Bwdl(addrl.addr0.addrl.dAG.dAH.dBG.dBH) \
    v=*(short *)addr1; \
    dBH+= v; \
    dBG+= v; \
    dAG+= v+(vs=v<<1); \
    dAH-= v+(vs<<=1); \
    *(short *)addr3=(dBH+1)>>2; \
*(short *)addr0=(dBG+1)>>2;
```

Engineering: KlicsCode: CompPict: Backward.c

```
v=*(short *)addr2; \
    dBH= vs=v<<1: \
    dAH+= v+(vs=v<<1); \
    dAG+= v+(vs<<=1);
"define BwdE3:addr3.addr2.addr1.dAG.dAH.dBH) \
    v=*(shore *;addr3; \
    dAH+= v; \
    dAG+= "": \
    dBH-= v+(vs=v<<1); \
    dBH-= v+(vs<<=1); \
    *(short *)addrl=(dAH+1)>>2; \
    *(short *)addr2=(dAG+1)>>2; \
    "(short ")addr3=dBH>>1;
#define Bwd(base.end.inc) \
    addr0=base: \
    addr3=addr0-(inc>>2); \
    addr2=addr3-(inc>>2); \
    addrl=addr2-(inc>>2); \
    BwdS0(addr0.dAG.dAH.dBH); \
    addrl+=inc; \
    BwdS1(addr1,addr0,dAG,dAH,dBH); \
    addr2+*inc; \
while(addr2<end) { \</pre>
        Bwd2 (addr2.dAG,dAH,dBG,dBH); \
        addr3+=inc: \
        Bwd3 (addr3.addr2.addr1.dAG.dAH.dBG,dBH); \
        addr0+=inc; \
        Bwd0 (addr0, dAG, dAH, dBG, dBH); \
        addrl+=inc; \
        Bwdl(addrl,addr0.addrl,dAG,dAH,dBG,dBH); \
        addr2+=inc; \
    BwdE2(addr2.dAG,dAH,dBH); \
    addr3+=inc; \
    BwdE3 (addr3, addr2, addr1, dAG, dAH, dBH);
#define BwdS0r2(addr0,dAG,dAH,dBH) \
    v=*(snort *)addr0; \
    dAG= 0: \
    daH= v: \
    dBH= v; \
#define BwdSlr2(addr1.addr0.dAG.dAH.dBH) \
    v=*(short *)addrl: \
    dBH+= v>>2: \
    dAG+= v; \
dAH-= v<<1: \
    *(short *)addr0=dBH;
#define Bwd2r2(addr2,dAG,dAH,dBG,dBH) \
    v=*(short *)addr2; \
    dBG= 0; \
dBH= V; \
    dAH+= V; \
    dAG+= v<<1;
#define Bwd3r2(addr3.addr2.addr1.dAG.dAH.dBG.dBH) \
    v=*(short *)addr3; \
    dAH+= 0; \
dAG+= V; \
    dBG+= v: \
```

_ Engineering:KlicsCode:CompPict:Backward.c

```
dBH-= v<<1; \
    ' short ':addrl=dAH>>1;
    * short *!addr2*dAG>>1;
*define Bwd0r2(addr0.dAG.dAH.dBG.dBH) \
    v=*!short *!addr0: \
dAG= 0: \
    dAH= v: \
    dBH+= v: \
    dBG-= /<<1;
=define Bwdlr2(addrl.addr0.addr3.dAG.dAH.d9G.dBH) \
    v=*(short *)addrl; \
    dBH+= 0; \
    dBG+= v: \
    dAG+= V; \
    dAH-= v<<1; \
    *(short *)addr3=dBH>>1; \
*(short *)addr0=dBG>>1;
#define BwdE2r2(addr2.dAG,dAH,dBH) \
    v=*(short *)addr2; \
    dBH= v: \
dAH+= v: \
    dAG+= v<<1;
=define BwdE3r2(addr3,addr2,addr1,dAG,dAH.dBH) \
    v="(short ")addr3; \.
    dAH+= 0; \
    dAG+= V: \
    dBH== v; \
dBH== v<<1; \
'(short ')addrl=dAH>>1; \
'(short ')addr2=dAG>>1; \
'(short ')addr3=dBH;
*define Bwdr2 | base, end, inc) \
    addr0=base; \
addr3=addr0-(inc>>2); \
    addr2=addr3-(inc>>2); \
    addrl=addr2-(inc>>2); \
    BwdS0r2(addr0,dAG,dAH,dBH); \
    addrl+=inc: \
    BwdS1r2(addr1,addr0,dAG,dAH,dBH): \
    addr2+=inc; \
while(addr2<end) ( )
         Bwd2r2(addr2.dAG,dAH,dBG,dBH); \
         addr3+=inc: \
         Bwd3r2(addr3.addr2.addr1.dAG,dAH,dBG,dBH); \
         addr0+=inc: \
         Bwd0r2(addr0.dAG.dAH.dBG.dBH); \
         addrl+=inc; \
         Bwdlr2(addr1,addr0,addr3,dAG,dAH,dBG,dBH); \
         addr2+=inc: \
     BwdE2r2(addr2,dAG,dAH,dBH); \
     addr3+=inc: \
     BwdE3r2(addr3,addr2,addr1,dAG,dAH,dBH):
 #define BwdS0r3(addr0.dAG,dAH,dBH) \
     v= (short *)addr0; \
     dAG= 0; \
     dAH= 0: \
```

```
Engineering:KlicsCode:CompPict:Backward.c
    d9H= v>>1: \
*define BwdSlr3(addr1.addr0.dAG,dAH,dBH) \
    v=*(short *)addrl: \
    dBH+= V>>3: \
    dAG+= V; .
    dAH-= v; \
    *(short *)addr0=dBH<<1;
#define Bwd2r3(addr2.dAG.dAH.dBG.dBH) \
    v=*(short *)addr2; \
    dBG= 0: \
    dBH= 0: \
    dAH+= v: \
    dAG+= v;
#define Bwd3r3(addr3.addr2.addr1.dAG.dAH.dBG,dBH) \
    v=*(short *)addr3; \
    dAH+= 0: \
    dAG+= 0: \
    dBG+= v: \
    dBH-= v; \
'(short ')addrl=dAH; \
'(short ')addr2=dAG;
#define Bwd0r3(addr0,dAG,dAH,dBG,dBH) \
    v=*(short *)addr0; \
    dAG= 0; \
dAH= 0; \
    dBH+= V; \
    dBG+= V;
#define Bwdlr3(addr1.addr0.addr3.dAG.dAH.dBG.dBH) \
    v=*(short *)addr1; \
    dBH+= 0; \
    dBG+= 0; \
    dAG+= v; \
    dAH-= v; \
    *(short *)addr3=dBH; \
*(short *)addr0=dBG;
*define BwdE2r3(addr2.dAG.dAH.dBH) \
    v=*(short *)addr2: \
    dBH= v>>1; \
dAH+= v; \
    dAG+= V;
#define BwdE3r3(addr3,addr2,addr1,dAG,dAH,dBH) \
    v=*(short *)addr3; \
    dAH+= 0; \
    dAG+= 0; \
    dBH-= v; \
    dBH-= v; \
*(short *)addrl=dAH; \
*(short *)addr2=dAG; \
*(short *)addr3=dBH<<1;
#define Bwdr3(base.end.inc) \
    addr0=base; \
addr3=addr0-(inc>>2); \
    addr2=addr3-(inc>>2); \
    addrl=addr2-(inc>>2); \
    BwdS0r3(addr0,dAG,dAH,dBH); \
```

```
Engineering: KlicsCode: CompPict: Backward.c
    addrl+=inc: \
    BwdSlr3(addr1.addr0.dAG,dAH.dBH): . .
    addr2+=inc: \
    while(addr2<end) {
         Bwd2r3(addr2.dAG.dAH.dBG.dBH); \
         addr3-=inc: \
         Ewd3r3:addr3.addr2.addr1.dAG.dAH.dBG.dBH); \
         addr0-=inc: \
         Bwd0r3(addr0, dAG, dAH, dBG, dBH); \
         addrl-=inc: \
         Bwdlr3(addr1,addr0,addr3,dAG,dAH,dBG,dBH); \
         addr2+=inc: \
    BwdE2r3(addr2.dAG.dAH.dBH); \
    addr3+=inc: \
    BwdE3r3(addr3,addr2,addr1,dAG,dAH,dBH);
extern void FASTBACKWARD(char *data, long incl. long loop1, long inc2, char *end2) extern void HAARBACKWARD(char *data, long incl. long loop1, long inc2, long loop2)
extern void HAARTOPBWD(char *data.long height,long width);
/* extern void HAARXTOPBWD(char *data.long area);*/
         FasterBackward(char *data, long incl, long endl, long inc2, char *end2)
void
    register short v, vs. v3. dAG, dAH. dBG, dBH. inc: register char 'addr0, 'addr1, 'addr2, 'addr3, 'end:
     register char
              'base;
    char
     inc=inc1:
     for(base=data:base<end2;base+=inc2) (</pre>
          end=base+endl;
         Bwd(base, end, inc);
}
                   TOPBWD(char *data, char *dst. long size_1, long size_0);
extern void
         TestTopBackward(short *data.int size(2).int cct_src)
              oct, area=size(0)*size(1)<<1;
     int
              width=size(0)<<1;
     short
               *top=area+(char *)data, *left=width+(char *)data:
     char
     for(oct=oct_src-1:oct>0:oct--) (
                   cinc=2<<oct, cinc4=cinc<<2.
          long
                   rinc=size(0)<<oct+1. rinc4=rinc<<2; /* col and row increments in t
          FASTBACKWARD((char *)data,rinc4,area-(rinc<<1),cinc.left):
          FASTBACKWARD((char *)data,cinc4.width-(cinc<<1),rinc.top);
 /* FasterBackward((char *)data,size[0]<<3,area-(size[0]<<2),2,left);
FasterBackward((char *)data,8,width-4,size[0]<<1,top);*/
TOPBWD((char *)data,(char *)data,size[0],size[1]);</pre>
          TestBackward(data, size, oct_src)
 void
 short *data:
 ınt
          size[2], oct_src;
               oct, area=size(0)*size(1)<<1;
      int
              width=size(0)<<1;
      short
               *top=area+(char *)data, *left=width+'char *)data;
     char
```

```
Engineering: KlicsCode: CompPict: Backward.c
    for(cct=oct_src-l:oct>=0:oct--) {
                cinc=2<<cct. cinc4=cinc<<2.
        long
                 rinc=Size(0)<<oct+1. rinc4=rinc<<2: /* col and row increments in t
        FasterSackward((char *)data.rinc4.area-(rinc<<1).cinc.left);</pre>
        FasterBackward()cnar *)data.cinc4.width-(cinc<<1),rinc.top);
vord
       Backward3511(data.size.oct_src)
short 'data;
int
       size(2), cct_src;
    int
            oct. area=size(0)*size(1)<<1;</pre>
    short width=size(0)<<1;
    char
            *top=area+(char *)data, *left=width+(char *)data;
    for(oct=oct_src-1:oct>0:oct--) {
              cinc=2<<oct. cinc4=cinc<<2.
                 rinc=size[0]<<oct+1, rinc4=rinc<<2; /* col and row increments in t</pre>
        BACK3511((char *)data,rinc4.area-(rinc<<1).cinc.left);</pre>
        BACK3511((char *)data.cinc4.width-(cinc<<1).rinc.top);
    BACK3511V((char *)data.size(0)<<3.area-(size(0)<<2).4,left);
BACK3511H((char *)data.8,width-4,size(0)<<1.top);
    TOPBWD((char *)data,(char *)data,size[1],size[0]);*/
```

```
Engineering: KlicsCode: CompPict: Backward.a
```

```
© Copyright 1993 KLICS Limited All rights reserved.
   Mritten by: Adrian Lewis
.....
   680X0 3511 Backward code
   Coeffs 11 19 5 3
   become 3 5 1 1
                  'klics'
      363
       macro
       BwdStart0
                  £addr0, £dAG, £dAH, £dBH
                   (LaddrO), EdAH
       move.w
                                  : dAH=*(short *)addr0
                                  ; dAG=v
       move.w
                   DAD4,HAL4
                   £dAG
       neg.w
                                  ; dAG= -dAG
       move.w
                   &dAH,&dBH
                                   ; dBH=v
       add.w
                   Edbh. Edbh
                                  : dBH=v<<1
     enda
       macro
       BwdStart1
                   &addrl.&addr0.&dAG.&dAH.&dBH
       move.w
                   (Laddrl), d0
                                  : v=*(short *)addrl
       move.w
                   d0,d1
                                   ; V5=V
       asr.w
                   *1,d1
                                   : vs=v>>1
       add.w
                   dl. &dBH
                                   ; dBH+= V>>1
       add.w
                   d0.&dAG
                                  ; dAG+=v
       sub.w
                   dO, &dAH
                                  ; dAH-=v
       add.w
                   d0.d0
                                  ; v<<=1
       add.w
                   d0. &dAG
                                  ; dAG-=2v
       add.w
                  d0.d0
                                  ; v<<=l
                  40,6dAH
       sub.w
                                  : dAH-=4v
       asr.w
                                  ; dBH>>=i
       move.w
                  &dBH.(&addr0)
                                 : *(short *)addr0=dBH
       endm
       macro
       BwdEven &addr2.&dAG,&dAH,&dBG,&dBH
       move.w
                   (&addr2),d0
                                  ; v=*(short *)addr2
       move.w
                  d0.&dBH
                                  ; dBkav
                                  ; dBG=v
       move.w
                  dO. EdBG
                                  : dBG=-v
                  &dBG
       neg.w
       add.w
                  HAD&. 0b
                                  ; dAH+=v
       add.w
                  dO, &dAG
                                  : dAG+=v
                  40,40
d0,40
d0,40
d0,40
       add.w
                                  ; 2v
       add.w
                                  ; dAH+=v
       add.w
                  40.40
                                  ; 2v
       add.w
                  do, &dAG
                                  ; dAH+=v
       endm
       Dacke
```

Engineering: KlicsCode: CompPict: Backward.a

```
3wdOdd
             &addr3.&addr2,&addr1,&dAG,&dAH,&dBG,&dBH
move.w
             (&addr3),d0
                              : v=*!short *laddr3
add. w
             d0. Edan
                              : dAH+=v
add.w
             DADA, Ob
                              : dAG+=v
add.w
             dC, &dBG
                              : dBG+=v
sub.w
             10,&dBH
                              : dBH-=v
add.w
             d0.d0
                              ; 2v
add.w
             d0,&dBG
                              ; dBG+=v
add.w
             d0.d0
                              : 4v
sub.w
             Habs, Ob
                              ; dBH-=4v
AST.W
             #2.&dAH
                              ; dAH>>=2
move.w
             &dAH, (&addrl)
                              ; *(short *)addrl=dAH
             12. EdAG
                              ; dAG>>=2
asr.w
move.w
                              ; *(short *)addr2=dAG
             &dAG, (&addr2)
endm
macro
awdEnd2
             &addr2,&dAG,&dAH,&dOH
move.w
             (&addr2).d0
                              ; v=*(short *)addr2
add.w
             MADA. 0D
                              : dAH+=v
add.w
             d0, EdAG
                              ; dAG+=v
add.w
             d0, d0
                              ; 2v
move.w
            d0, EdBH
                              ; dBH=2v
add.w
             do, edah
                              ; dAH+=2v
add.w
             0b,0b
add.w
             do, Lang
                              ; dAG+=4v
endn
BwdEnd3
             %addr3, &addr2, &addr1, &dAG, &dAH, &dBH "
             (&addr3),d0
move.w
                              ; v=*(short *)addr3
            do, Edah
do, Edag
add.w
                               dAH+=v
add.w
                              ; dAG+=v
lsl.w
            #3,d0
                              ; 8v
sub.w
            d0.&dBH
                              ; dBH-=8v
             #2.&dAH
asr.w
                               dAH>>=2
move.w
             &dAH.(&addrl)
                               *(short *)addrl=dAH
asr.w
             #2. SdAG
                               dAG>>=2
move.w
             &dAG, (&addr2)
                              ; *(short *)addr2=dAG
asr.w
             *1,&dBH
                              ; dBH>>=1
move.w
             EdBH, (&addr3)
                             ; *(short *)addr3=dBH
euqu.
macro
Bwd
            &base, &end, &inc
movea.1
                                      ; addr0=base
            Lbase, a0
move.1
            &inc.d0
                                      ; d0=inc
asr.l
             #2,d0 .
                                      ; d0=inc>>2
movea.1
            a0.a3
                                      ; addr3=addr0
suba.l
            d0, a3
                                      ; addr3-=(inc>>2)
                                      ; addr2=addr3
movea.l
            a3, a2
suba.l
            d0, a2
                                      ; addr2-=(inc>>2)
movea.1
            a2.a1
                                      : addrl=addr2
```

Engineering: KlicsCode: CompPict: Backward.a

```
suba.l
                     d0.al
                                              : addrl-=(inc>>2)
        BwdStartC
                     a0.d4.d5.d7
                                              : 3wdStart3(addrC,dAG,dAH,dBH)
        adda.l
                     &inc.al
                                              : addrl+=inc
        3wdStart1
                     al.a0.d4.d5,d7
                                              : BwdStartl(addrl,addr0,dAC,dAH.dBH)
        adda.l
                     &inc.a2
                                                addr2+=inc
?do
        3wdEven
                     a2.d4.d5.d6.d7
                                              : BwdEven(addr2.dAG,dAH,dBG.dBH)
        adda.1
                     inc.ai
                                                addr3+=inc
        3wd0dd
                     a3.a2.a1.d4.d5,d6.d7
                                              : BwdOdd(addr3,addr2,addr1,dAG,dAH,dBG
        adda.l
                    &inc.a0
                                                addr0+=inc
        BwdEven
                    aC.d6.d7.d4.d5
                                                BwdEven(addr0, dBG, dBH, dAG, dAH)
        adda.1
                                              ; addrl+=inc
                     &inc,al
        bb0bw6
                    al.a0,a3.d6,d7.d4.d5
                                              ; BwdOdd(addr1,addr0,addr3,dBG,dBH.dAG
        adda.l
                     &inc.a2
                                              : addr2+=inc
        cmpa.1
                     a2.&end
                                              ; addr2<end
        bgt.s
                    @do
                                              ; while
        BwdEnd2
                     a2,d4,d5,d7
                                              ; BwdEnd2(addr2,dAG,dAH,dBH)
        adda.l
                     £inc.a3
                                              ; addr3+=inc
        BwdEnd3
                     a3, a2, a1, d4, d5, d7
                                              ; BwdEnd3(addr3.addr2,addr1.dAG,dAH.dB
        endm
Back3511 FUNC
                     EXPORT
?S
        RECORD
data
        DS.L
                     1
        DS.L
inci
                     1
endl
        25.L
                     1
inc2
        DS.L
                     1
end2
        DS.L
                     1
        ENDR
        link
                     a6,#0
                                             ; no local variables
                    d4-d7/a3-a5,-(a7)
        movem.l
                                              ; store registers
        move.1
                     PS.incl(a6),d3
                                              : inc=incl
                    PS.data(46),45
        movea.1
                                              ; base=data
8do
        movea.l
                    a5, a4
                                              ; end=base
        adda.l
                     PS.endl(a6),a4
                                              : end+=end1
        Bvd
                     a5,a4,d3
                                              ; Bwd(base,end,inc)
        adda. 1
                    PS.inc2(a6),a5
                                              ; base+=inc2
        cmpa.1
                    PS. end2 (a6), a5
                                              : end2>base
        blt.w
                    9do
                                              ; for
        movem. 1
                    (a7) + d4 - d7/a3 - a5
                                            ; restore registers
        unik
                                              : remove locals
        rts
                                             ; return
        ENDFUNC
        macro
        BwdStartVO &addrO, &dAG, &dAH, &dBH
                     (&addr0), &dAH
        move.l
                                     ; dAH=*(short *)addr0
        move.1
                    Edan, Edag
                                       dAGev
        neg.l
                    &dAG
                                     : dAG= -dAG
                    Edan, Edbn
                                     ; dBH=v
        move.1
        add.l
                    &dBH. &dBH
                                     ; dBH=v<<1
        endn
                  ------
        macro
```

BwdStartV1 &addr1,&addr0,&dAG,&dAH,&dBH

```
=Engineering: KlicsCode: CompPict: Backward.a
                             ; y=*(short *)addrl
move. 1
             (&addrli.d0
move.1
            d0.d1
                              ; 75=7
                             ; 75=V>>l
asr.l
            #1.dl
            d1.4dBH
                              ; dBH+= v>>1
add.l
            d0.&dAG
                              : dAG+=V
add. 1
             d0, &dAH
                             : dan-=v
sub. l
add.1
            d0.d0
                              ; v<<=l
                             : dAG+=2v
add.l
            d0,4dAG
add.1
             d0,d0
                             ; v<<=l
             d0,&dAH
                              : dAH-=4V
sub. 1
asr.l
             #1,&dBH
                              ; dBH>>=1
add.w
             &dBH.&dBH
                             ; shift word back
             #1,&dBH
                              ; dBH>>=1
asr.w
             &dBH.(&addr0)
                             ; *(short *)addr0=dBH
move.l
endm
macro
             &addr2,&dAG,&dAH,&dBG,&dBH
BwdEvenV
                              ; v=*(short *)addr2
move.1
             (&addr2),d0
                              : dBH=v
             d0,&dBH
move.i
                              ; dBG=v
             do, &dBG
move. 1
                              : dBG=-v
             &dBG
neg.1
                              : dAH+=v
             HADA.Ob
add.1
                              ; dAG+=V
add.l
             d0.&dAG
add.l
             0b.0b
                              ; 2v
                              ; dAH+=v
add. 1
             d0.4dAH
add.l
             d0.d0
                              ; 2v
             do, Edag
                              ; dAH+=V
add.1
endm
macro
             &addr3, &addr2, &addr1, &dAG, &dAH, &dBG, &dBH
BwdOddV
                              ; v=*(short *)addr3
move.l
             (&addr3).d0
                              ; dAH+=V
add.l
             d0.&dAH
                              ; dAG+=v
add.l
             d0,&dAG
                              ; dBC+=v
add.l
             d0, &dBG
             do. &dBH
                              : dEH-=v
sub. 1
             d0,d0
                              ; 2v
 add.l
                              ; dBG+=v
             dO, LdBG
 add. 1
                              ; 4v
             d0.d0
 add.l
             d0,&dBH
                              ; dBH-=4v
 sub. l
             #2,&dAH
                              ; dAH>>=2
 asr.l
                              ; shift word back
             #2,&dAH
 isl.w
              #2,&dAH
                              : dAH>>=2
 asr.w
                              ; *(short *)addrl=dAH
              &dAH. (&addr1)
 move.1
                              ; dAG>>=2
 asr.l
              #2, £dAG
                              ; shift word back
             #2, £dAG
 isl.w
                              ; dAG>>=2
 asr.w
              #2, £dAG
                              : *(short *)addr2=dAG
              &dAG, (&addr2)
 move.l
 endm
 macro
 BwdEndV2
              &addr2,&dAG,&dAH,&dBH
```

(saddr2).d0

move 1

: v==(short *)addr2

ı

```
Engineering: KlicsCode: CompPict: Backward.a
```

```
v=+HAb :
                   HADA. 0b
      add.l
                   do.&dAG
                                    · dAG+=7
      acc.l
      add.1
                   00,00
                                    : 2v
                   d0.&dBH
                                    : dBH=2v
      move.1
                                    : dAH+=2v
      add.i
                   d0.&dAH
                                    : 4v
      add.:
                   d0.d0
      add.!
                   de.&dAG
                                    : dAG+=4V
       endm
      macro
                   &addr3, &addr2, &addr1, &dAG, &dAH, &dBH
       BwdEndV3
                                    ; v=*(short *)addr3
                   (&addr3).d0
       move.l
                   HAD&, OD
                                    # dAH+=V
       add.l
                   DAD 3.0b
                                    ; dAG+=V
       add.l
       151.1
                    #3.d0
                                    : 8v
                                    ; dBH-=8v
                   MEDA, Ob
       sub. l
                    #2,&dAH
                                    ; dAH>>=2
       asr.l
                                    ; shift word back
                    #2,&dAH
      .lsl.w
                                    ; dAH>>=2
                    #2,&dAH
       asr.w
                                    : *(short *)addrl=dAH
       move.1
                    &dAH.(&addrl)
                                    ; dAG>>=2
                    #2,&d&G
       asr.l
                                    ; shift word back
                    #2.&dAG
       1s1.w
                                    ; dAG>>=2
                    #2, &dAG
       asr.w
                   &dAG.(&addr2)
                                    ; *(short *)addr2=dAG
       move.1
                                    ; dBH>>=1
       asr.l
                                    : shift word back
                    #1.&dBH
       lsl.w
                                    ; dAH>>=2
                    #1.&dBH
       asr.w
                                     ; dBH<<=1
                    Edby, Edby
       add.l
                                   ; *(short *)addr3=dBH
                    &dBH,(&addr3)
       move.1
       endm
       macro
                    &base,&end,&inc
       BwdV
                                             ; addr0=base
       movea.1
                    &base.a0
                                             ; d0=inc
       move.1
                    &inc.d0
                                             ; d0=inc>>2
        asr.l
                    +2.d0
                                             ; addr3=addr0
       movea.1
                    a0,a3
                                             ; addr3-=(inc>>2)
        suba.1
                    d0.a3
                                             : addr2=addr1
                    a3.a2
        movea.1
                                             ; addr2-=(inc>>2)
                    d0, a2
        suba.:
                                             ; addrl=addr2
                    a2.a1
        movea.l
                                             ; addr1-=(inc>>2)
                    d0.al
        suba. 1
                                             ; BwdStart0(addr0,dAG,dAH,dBH)
        BwdStartVO a0.d4.d5.d7
                                             ; addrl+=inc
                    &inc.al
        adda.l
                                             ; BwdStartl(addrl.addr0.dAG,dAH,dBH)
        BwdStartV1 al.a0,d4,d5.d7
                                             ; addr2+=inc
        adda.l
                    &inc.a2
                                             ; BwdEven (addr2, dAG, dAH, dBG, dBH)
                    a2,d4,d5,d6,d7
€do
        BwdEvenV
                                               addr3+=inc
        adda.l
                    &inc,a3
                                             ; BwdOdd(addr3,addr2,addr1,dAG,dAH,dBG
                    a3.a2.a1.d4.d5.d6.d7
        BwdOddV
                                             ; addr0+=inc
        adda.l
                    &inc.a0
                                             ; BwdEven(addr0,dBG,dBH,dAG,dAH)
                    a0,d6,d7,d4.d5
        BwdEvenV
                                             ; addrl+=inc
        adda.l
                    &inc,al
                                             ; BwdOdd(addrl.addr0.addr3.dBG,dBH.dAG
                    al.a0.a3,d6.d7,d4.d5
        BwdOddV
                                             ; addr2+=inc
        adda.l
                    &inc.a2
                                             ; addr2<end
                    a2, £end
        cmpa.1
                                             ; while
                    ಯಾ
        bgt.s
                                             ; BwdEnd2 (addr2.dAG.dAH.dBH)
        BwdEndV2
                    a2,d4,d5,d7
                                             ; addr3-=inc
        adds.l
                    Sinc.a3
```

move.1

move.w

move.w

move.w

move.w

SWAD

swap

£B, d2

d1.d2

4B,d1

40, £B

d1.d0

£B **d0**

; B=3G1H, d0=3H1G, d1=5H1G, d2=1H5G : B=3G1H, d0=1G3H, d1=5H1G, d2=1H5G

```
Engineering: KlicsCode: CompPict: Backward.a
           EvdEndV3
                             a3, a2. a1, d4, d5, d7
                                                               ; BwdEnd3(addr3.addr2.addr1,dAG,dAH,dB
           encin
Back3511V FUNC
                             EXPORT
25
           RECORD
                             6
data
           DS.L
                             1
incl
           DS.L
                             1
endl
           DS.L
                             1
inc2
           DS.L
                             1
end2
            DS.L
            ENDR
                             a6,#0
                                                                 ; no local variables
            link
           movem.1
                             d4-d7/a3-a5,-(a7)
                                                                 ; store registers
                             PS.incl(a6).d3
            move.l
                                                                 ; inc=incl
                             PS.data(a6),a5
                                                                : base=daca
            movea.l
                                                                ; end=base
340
            movea.1
                             a5,a4
                             PS.endl(a6),a4
                                                                ; end+=endl
            adda.l
            BwdV
                              a5,a4,d3
                                                                ; Bwd(base, end, inc)
            adda.1
                             PS.inc2(a6),a5
                                                                : base+=inc2
                              PS. end2 (a6) .a5
                                                                 ; end2>base
            cmpa.1
                              9ರೂ
                                                                 ; for
            ble.v
                                                                ; restore registers ; remove locals
                              (a7)+,d4-d7/a3-a5
            movem.l
            unlk
                              a6
            rts
                                                                 : return
            ENDFUNC
            macro
            BwdStartH
                             &addrR, &A, &C
                                                  ; 1H1G=*(long *)addrR
                              (EaddrR)+,&A
            move.1
                                                    ; A=1H1G, d0=1H1G
                              ٤٨, d0
            move. L
                                                    ; A=1H1G, d0=1H1G;
; A=1H1G, d0=1H2G, C=1H1G;
; A=1H1G, d0=1H2G, C=1H1G;
; A=1H3G, d0=1H2G, C=1H1G;
; A=1H3G, d0=1H5G, C=1H1G
                              EA, EC
            move.1
                              0b, A&
            add.w
                              d0, £A
            add.w
            add.w
                              EA, dO
                                                    : A=3GH1, d0=1H5G, C=1H1G
; A=AAAA, d0=1H5G, C=1H1G
            swap
                              éλ
            sub.1
                             do, LA
                          -----
            macro
            BwdCycleH &addrR.&addrW.&A.&B.&C
                                                  ; 1H1G=*(long *)addrR
                              (&addrR)+,&B
            move. 1
                                                  ; lHlG=*(long *)addxR

: B=1H1G, d0=1H1G

; B=1H1G, d0=2H2G

: B=1H1G, d0=2H2G, d1=2H2G

; B=1H1G, d0=3H3G, d1=2H2G

; B=1H1G, d0=3H3G, d1=5H5G, d2=1H1G

; B=1H1G, d0=3H3G, d1=5H5G, d2=1H5G

; B=1H1G, d0=3H3G, d1=5H1G, d2=1H5G

; B=1H3G, d0=3H3G, d1=5H1G, d2=1H5G

; B=1H3G, d0=3H1G, d1=5H1G, d2=1H5G

; B=1H3G, d0=3H1G, d1=5H1G, d2=1H5G

; B=1G1H, d0=3H1G, d1=5H1G, d2=1H5G
            move.1
                              &B.40
                              40.40
            add.1
            move.1
                              d0,d1
                              05,g3
            add.l
            add.l
                              40,41
```

Engineering:KlicsCode:Comp@ict:Backward.a

```
d2.4B
                                       : B=3G1H-1H5G
        sub. 1
                     d0. &A
        add.l
                                      ; A+=1H3G
        add. l
                     d1.6A
                                      ; A+=5G1H
                                      : A0>>=2
                     42. SA
        w.zs
       move. w
                     &A. SC
                                      : C complete
                     +2. £A
                                      ; Al>>=2
        asr. :
                     &C. (SaddrW) +
                                      ; *(long *)addrW=DD
       move. 1
                                      ; C=AlXX
                     EA. £C
        move.:
       macro
                     &addrR. &addrW, &A, &B, &C
        BwdEndH
                                      ; lHlG=*(long *)addrR
        move.1
                     (&addrR)+,d0
                                      ; d2=1G
        move.w
                     d0.d2
                                      ; d2=4G
                     #2.d2
        Isl.w
                                       : d2=-4G
                     đ2
        neg.w
                                      ; d0=1G1H
        swap
                     d0
                                       ; d2+-1H
        add.w
                     d0,d2
                                       ; d0=1G1H, d1=1G1H
                     d0.d1
        move.1
                                       ; d0=1G1H, d1=1G2H
; d0=1G3H, d1=1G2H
        add.w
                     d0.d1
        add.w
                     d1.d0
                                       : d0=1G3H, d1=1G5H
: d0=1G3H, d1=5H1G
                     d0.d1
        add.w
        swap
                     d1
                                       ; A+=1G3H
        add.l
                     do, EA
        add.1
                     dl. &A
                                      ; A+=5H1G
                     42, 4A
                                      ; Al>>=2
        asr.w
                                       ; C complete
                     &A. &C
        move.w
                     42. &A
                                       ; A0>>=2
        asr.l
                     EC. (EaddrW) +
                                       : *(long *)addrW=C
        move.l
                     d2,&A
                                       ; A=D1D2
        move.w
                                      : *(long *)addrW=A
                     EA. (EaddrW) +
        move.1
        endm
        macro
        BwdH
                     &base.&end.&inc
                                             : addrR=base
                     &base.a0
        movea.1
                                               : addrw=addrR
        movea.1.
                     a0,al
                                               ; BwdStart(addrR.A.DD)
                     aC.d3.d5
        BwdStartH
                                               ; BwdCycle(addrR.addrW,A,B,C)
                     a0,a1.d3.d4.d5
obs
        BwdCycleH
                                               ; BwdCycle(addrR.addrW.B.A.C)
                     a0.a1.d4.d3.d5
a0.&end
        BwdCycleH
                                               : addr2<end
        cmpa.1
                                               ; while
                     @do
        bgt.s
                                               ; BwdEnd(addrR.addrW, A, B, DD)
        BwdEndH
                     a0.a1.d3.d4.d5
        endm
Back3511H FUNC
                     EXPORT
PS
        RECORD
                     8
        DS.L
data
incl
        DS.L
                     1
endl
        DS.L
inc2
        DS.L
end2
        DS.L
        ENDR
                                               : no local variables
                     36. EA
        link
```

END

- 791 -

	Engineering: KlicsCode: CompPict: Backward.a			Page 3
•	movem.l	d4-d7/a3-a5(a7)	; store registers	
	move.l	PS.incl(a6).d3	; inc=incl	
	movea.l	PS.data(a6),a5	; Dase=data	
3do	movea.l	a5.a4	; end=base	
	adda.:	PS.endl(a6).a4	; end+=endl	
	9wdH	a5,a4,d3	<pre>; Bwd(base.end.inc)</pre>	
	adda.l	PS.inc2(a6),a5	; base+=inc2	
	cmpa.l	PS.end2(a6),a5	: end2>base	
_	blc.w	0 0	; for	
•	movem.1	(a7)+,d4-d7/a3-a5	; restore registers	
	unlk	a6	: remove locals	
	rts		; return	
•	ENDFUNC			

Engineering:KlicsCode:CompPict:KlicsEnc.c

```
© Copyright 1993 KLICS Limited
 · All rights reserved.
   Written by: Adrian Lewis

    Full still/video Knowles-Lewis Image KlicsEncode System utilising HVS properti

   and delta-tree coding
    Recoded and re-rationalised (Stand alone version)
*include
              <PixMach.h>

≠include
            Bits3.h.
*include
              'Klics.h'
*include
              'KlicsHeader.h'
•include
              'KlicsEncode.h'
*include
              <Math.h>
/* If bool true the negate value */
*define negif(bool, value) ((bool)?-(value):(value))
                                  negif(value<0, value)
*define abs(value)
extern void
                   HaarForward();
                   Daub4Porward();
excern void
/* Use the bit level file macros (Bits2.h)
buf_use; */
/* Huffman encode a block */
#define HuffEncLev(lev.buf) \
     HuffEncode(lev(0),buf); \
     HuffEncode(lev[1],buf); \
HuffEncode(lev[2],buf); \
     HuffEncode(lev[3], buf):
/* Fixed length encode block of integers */
*define IntEncLev(lev,lpf_bits,buf) \
     IntEncode(lev(0).lpf_bits.buf): \
IntEncode(lev(1).lpf_bits.buf): \
     IntEncode(lev(2), lpf_bits, buf); \
IntEncode(lev(3), lpf_bits, buf);
 /* Define write a zero */
*define Token0 \
   buf_winc(buf);
/* Define write a one */
*define Tokenl \
     buf_set(buf); buf_winc(buf);
/* Write block for data and update memory */
#define DoXfer(addr.pro.lev.dst.mode.cct.nmode.buf) \
     HuffEncLev(lev.buf); \
     PutData(addr.pro.dst); \
     mode(oct)=oct==0?M_STOP:nmode;
 " Function Name: Quantize
```

```
Engineering: KlicsCode: CompPict: KlicsEnc.c
    Description:
                    H.261 style quantizer
    Arguments: new, old - image blocks
                pro, lev - returned values
                q - quantizing divisor
   Returns:
                lev is all zero, quantized data (pro) & level (lev)
acclean Quantize(int new[4], int old[4], int pro[4], int lev[4], short q)
            blk, half_q=(1<<q)-1>>1;
    for(blk=0;blk<4;blk++) (
        int
                data=new(blk)-old(blk),
                mag_level=abs(data)>>q;
        mag_level=mag_level>135?135:mag_level;
        lev(blk)=negif(data<0,mag_level);</pre>
        pro(blk)=old(blk)+negif(data<0, (mag_level<<q)+(mag_level!=0?half_q:0));</pre>
    return(pro(0)==0 && pro(1)==0 && pro(2)==0 && pro(3)==0);
)
void
        QuantizeLPF(int new[4],int pro[4],int lev[4],short q)
            blk, half_q=(1<<q)-1>>1;
    for(blk=0:blk<4:blk++) (
        int
                data=new(blk),
                mag_level=abs(data)>>q;
        lev(blk)=negif(data<0.mag_level);</pre>
        pro(blk) = (lev(blk) << q) + half_q;</pre>
)
   Function Name: GuessOuantize
                   Estimate threshold quantiser value
   Description:
   Arguments: new, old - image blocks
                q - q weighting factor
   Returns:
                estimated q_const
float
       GuessQuantize(int new(4),int old(4),float q).
    int
            blk:
    float
           q:_max=0.0;
    for(blk=0;blk<4;blk++) (
               i, data=abs(new(blk)-old(blk));
        for(i=0:data!=0;i++) data>>=1;
        if (i>0) i--:
        qt = (((3 << i) - 1) >> 1)/q;
        qt_max=qt_max>qt?qt_max:qt;
    return(qt_max);
}
   Function Name: IntEncode
    Description:
                   Write a integer to bit file
    Arguments: lev - integer to write now signed
```

```
Engineering:KlicsCode:CompPict:KlicsEnc.c
                    bits - no of bits
         IntEncode(int lev.int bits. Buf buf)
void
" Old version
    int
    for(i=bits-1;:>=0::--) {
    if (lev&(1<<1); buf_set(buf);</pre>
         buf_winc(buf):
    ١
• /
/* New version
    int i, mag=abs(lev):
Boolean sign=lev<0:</pre>
     if (l<<bits-1 <= mag) mag=(l<<bits-1)-1;</pre>
     if (sign) buf_set(buf);
     buf_winc(buf);
     for(1=1<<bits-2;i!=0;i>>=1) (
          if (mag&i) buf_set(buf);
          buf_winc(buf):
     10/
/* Hardware compatable version: sign mag(lsb->msb) */
              i, mag=abs(lev);
     Boolean sign=lev<0:
     if (1<<bits-1 <= mag) mag=(1<<bits-1)-1;
if (sign) buf_set(buf);</pre>
     but_winc(but);
     for(i=1;i!=1<<bits-1;i<=1) (</pre>
          if (mag&i) buf_sec(buf);
          buf_winc(buf);
     )
1
/* Function Name: HuffEncedeSA
   Description: Write a Huffman coded integer to bit file
   Arguments: lev - integer value
   Returns: no of bits used
          HuffEncode(int lev.Buf buf)
bicv
/* int
              level=abs(lev):
     if (level>1) buf_set(buf);
     buf_winc(buf);
     if(level>2 () level==1) buf_set(buf);
     buf_winc(buf);
     if (level:=0) (
   if (lev<0) buf_set(buf);</pre>
          buf_winc(buf);
          if (level>2) (
               for(i=3:i<level;i++) (</pre>
                    buf_winc(buf);
               buf_set(buf);
               buf_winc(buf);
```

1.1

```
Engineering:KlicsCode:CompPict:KlicsEnc.c
```

```
/* New version */
            level=abs(lev), i:
    int
    if (level!=0) buf_set(buf);
    buf_winc(buf):
    if (level:=0) {
        if (lev<0) buf_set(buf);</pre>
        buf_winc(buf):
        if (level<8) (
             while (1<level--)
                 buf_winc(buf);
             buf_set(buf):
             buf_winc(buf):
        } else (
             for(i=0:i<7:i++)
                 buf_winc(buf);
             level-=8;
             for(i=1<<6;i!=0;i>>=1) (
                 if (level&i) buf_set(buf);
                 buf_winc(buf);
             }
        }
    )
)
    Function Name: KlicsEChannel
                     Encode a channel of image
    Description:
    Arguments: src - source channel memory
                 dst - destination memory (and old for videos)
octs, size - octaves of decomposition and image dimensions
                 normals - HVS weighted normals
                  lpf_bits - no of bits for LPF integer (image coding only)
 • /
         KlicsEncY(short *src,short *dst,int octs,int size(2),int thresh(5), int co
void
             oct, mask, x, y, sub, tmp, step=2<<octs, blk[4], mode[4], nz, no, base
     int
             addr(4), new(4), old[4], pro[4], lev(4), zero[a]=(0,0,0,0);
     int
     Boolean nzflag, noflag, origin;
             bitmask=-1<<kle->seqh.precision-kle->frmh.quantizer(0)-1;
     int
             buf=&kle->buf;
     Buf
     for(y=0;y<size(1);y+=step)
     for(x=0:x<size(0);x+=step)
     for(sub=0;sub<4;sub++) (
     mode(oct=octs-1)=base_mode;
     if (sub==0) mode[oct=octs-1] |= M_LPF;
     mask=2<<oct;
     do (
         GetAddr (addr, x,y, sub, oct, size, mask);
         gwitch(mode(oct)) (
         case M_VOID:
             GetData(addr,old,dst);
             else ( DoZero(addr.dst.mode.oct); }
break;
              if (Blk2ero(old)) mode(oct)=M_STOP;
         case M_SENDIM_STILL:
             GetData(addr, new, src);
              nz=Decide(new); nzflag=nz<=thresh(octs-oct);
              if (nzflag || Quantize(new.zero.pro.lev.kle->frmh.quantizer(octs-oct))
                  GetData(addr,old.dst);
```

```
Engineering: KlicsCode: CompPict: KlicsEnc.c
        if (BlkZero(old)) (
             Token0:
             mode(oct)=M_STOP:
        else (
             Tokenl: Tokenl:
             DoZero(addr,dst,mode.oct);
    e se (
        Token1: Token0:
        DoXfer(addr.pro.lev.dst.mode.oct.M_SENDIM_STILL.buf);
    break:
case M_SEND:
    GetData(addr,new,src);
    GetData(addr,old,dst);
    nz=Decide(new): nzflag=nz<=thresh(octs-oct);</pre>
    if (BlkZero(old)) (
        if (nzflag || Quantize(new,zero,pro,lev,kle->frmh.quantizer(octs-o
             Token0:
             mode(oct)=M_STOP;
         ) else (
             Token1: Token0:
             DoXfer(addr.pro.lev.dst.mode.oct.M_SEND(M_STILL.buf):
    ) else (
                 oz=Decide(old), no=DecideDelta(new.old);
         ınt
         Boolean motion=(nz+oz)>>oct <= no: /* motion detection */
         no=DecideDelta(new.old); noflag=no<=compare(octs-oct);
         origin=nz<=no:
             if ((!noflag || motion) && !nzflag) ( /* was !noflag && !nzfl
             if (Quantize(new, origin?zero:old, pro, lev, kle->frmh.quantizer[o
                 Tokenl; Tokenl; Token0;
                 DoZero(addr.dst.mode.oct);
             } else {
    if (origin) {
        Token1; Token0;
        "far(addr.pro
                      Doxfer(addr.pro,lev.dst.mode,oct,M_SEND(M_STILL,buf);
                 } else {
                      Token1: Token1: Token1:
DoXfer(addr,pro,lev,dst,mode,oct,M_SEND,buf):
         ) else (
                  if ((motion () origin) && nzflag) ( /* was origin && nzfla-
                 Token1: Token1: Token0:
                 DoZero(addr.dst,mode.oct):
             ) else (
                 Token0:
                 mode (oct) = M_STOP;
         )
    break;
 case M_STILL:
     GetData(addr.new.src);
     nz=Decide(new); nzflag=nz<=thresh(octs-oct);</pre>
     if (nzflag || Quantize(new,zero,pro,lev,kle->frmh.quantizer(octs-oct))
         Token0:
         mode (oct ) = M_STOP;
     ) else (
         Token1:
         DoKier(addr.pro.lev.dst.mode.cot.M_STILL.bufl;
```

```
Engineering: KlicsCode: CompPict: KlicsEnc.c
            break:
        case M_LPFIM_STILL:
            GetData(addr.new.src);
             QuantizeLPF(new.pro,lev.kle->frmh.quantizer(0));
             VerifyData(lev(0), bitmask, tmp);
             VerifyData(lev(1),bitmask.tmp);
             VerifyData(lev(2),bitmask.tmp);
             VerifyData(lev(3).Ditmask.tmp);
             IntEncLev(lev,kle->seqh.precision-kle->frmh.quantizer(0),buf);
             PutData(addr.pro.dst):
             mcde(oct)=M_QUIT;
            break:
        case M_LPFIM_SEND:
            GetData(addr,new.src);
             GetData(addr,old,dst);
             no=DecideDelta(new,old): noflag=no<=compare(octs-oct):</pre>
             if (noflag) (
                 Token0;
             ) else (
                 Token1;
                 Quantize(new,old,pro,lev,kle->frmh.quantizer(0)); .
                 HuffEncLev(lev,buf);
                 PutData(addr.pro.dst);
             mode (oct ) =M_QUIT;
             break:
        switch(mode(oct)) (
        case M_STOP:
             StopCounters(mode.oct.mask.blk,x,y,octs);
             break:
        case M_QUIT:
             break;
         default:
             DownCounters (mode, oct, mask, blk);
             break:
    ) while (mode(oct)!=M_QUIT);
}
        KlicsEncUV(short *src,short *dst,int octs.int size(2),int thresh[5]. int c
void
             oct. mask. x, y, X, Y, sub, tmp, step=4<<octs, blk[4], mode[4], nz, no addr(4), new(4), old[4], pro(4), lev(4), zero(4)=\{0,0,0,0,0\};
    inc
    int
    Boolean nzflag, noflag, origin;
    int
             bitmask=-1<<kle->seqh.precision-kle->frmh.quantizer[0]-1;
    Buf
             buf=&kle->buf:
    for(Y=0;Y<size[1];Y+=step)
    for (X=0; X<size(0); X+=step;
    for(y=Y;y<size[1] && y<Y+step;y+=step>>1)
for(x=X;x<size[0] && x<X+step;x+=step>>1)
    for (sub=0; sub<4; sub++) {
    mode(oct=octs-1)=base_mode;
    if (sub==0) mode(oct=octs-1) i= M_LPF;
    mask=2<<oct:
    do (
         GetAddr(addr,x,y,sub.oct.size.mask);
         switch(mode(oct)) (
         case M_VOID:
             GetData(addr.old.dst):
```

```
Engineering: KlicsCode: CompPict: KlicsEnc.c
    if (BlkZero(old)) mode(oct)=M_STCP:
    else ( DoZero(addr,dst.mcde,oct;; )
    break:
case M_SENDIM_STILL:
    GetData(addr.new.src):
    nz=Decide(new); nzflag=nz<=thresh(octs-oct);</pre>
    if (nzilag ): Quantize(new.zero.pro.lev.kle->frmh.quantizer(octs-oct))
        GetData(addr.old.dst);
        if (BlkZero(old)) (
             Token0:
            mode(oct)=M_STOP;
        ) else {
            Tokenl: Tokenl:
            DoZero(addr.dst.mode.oct);
        ١
    ) else (
        Token1: Token0:
        DoXfer(addr.pro,lev.dst,mode.oct.M_SEND(M_STILL.buf);
    break:
case M_SEND:
    GetData(addr.new.src):
    GetData(addr.old.dst):
    nz=Decide(new); nzflag=nz<=thresh(octs-oct);</pre>
    if (BlkZero(old)) (
         if (nzflag || Quantize(new,zero,pro,lev,kle->frmh.quantizer(octs-o
             Token0:
             mode(oct)=M_STOP:
         } else (
             Token1: Token0:
             DoXfer(addr.pro.lev.dst.mode.oct.M_SEND(M_STILL.buf);
    } else (
                 oz=Decide(old), no=DecideDelta(new,old);
         int
         Boolean motion=(nz+oz)>>oct <= no: /* motion detection */
         no=DecideDelta(new,old): noflag=no<=compare(octs-oct):
         origin=nz<=no;
             if ((!noflag || motion) && !nzflag) ( /* was !noflag && !nzfl if (Quancize(new.origin?zero:old.pro.lev.kle->frmh.quantizer(o
                 Token1; Token1; Token0;
                 DoZero(addr.dst.mode.oct);
             ) else (
                 if (origin) (
                      Token1: Token0;
                      DoXfer(addr.pro.lev.dst.mode.oct.M_SEND(M_STILL,buf);
                  ) else (
                      Token1; Token1; Token1;
                      DoXfer(addr.pro,lev,dst,mode.oct,M_SEND.buf);
                 }
         } else {
                 if ((motion || origin) && nzflag) ( /* was origin && nzfla-
                 Token1: Token1: Token0:
                 DoZero(addr, dst, mode. oct);
             } else (
                 Token0:
                 mode(oct)=M_STOP;
             )
     break;
case M_STILL:
```

Engineering: KlicsCode: CompPict: KlicsEnc.c

```
GetData(addr.new.src):
            nz=Decide(new); nzflag=nz<=thresh(octs-oct);
            if (nzflag || Quantize(new,zero.pro.lev.kle->frmh.quantizer(octs-oct);
                 Token0:
                mode (oct ) =M_STOP;
            ; else (
                 Token1:
                 DoXfer(addr.pro.lev.dst.mode.oct.M_STILL.buf);
            break:
       case M_LPFIM_STILL:
            GetData(addr.new.src);
            QuantizeLPF(new,pro.lev,kle->frmh.quantizer(0]);
            VerifyData(lev(0),bitmask.tmp);
            VerifyData(lev(1), bitmask.tmp);
            VerifyData(lev[2],bitmask.tmp);
            VerifyData(lev(3), bitmask, tmp);
            IntEncLev(lev,kle->seqh.precision-kle->frmh.quantizer(0),buf);
            PutData(addr.pro.dut);
            mode(oct)=M_QUIT;
            break:
        case M_LPFIM_SEND:
            GetData(addr.new, src);
            GetData(addr.old.dst);
            no=DecideDelta(new,old); noflag=no<=compare(octs-oct);
             if (noflag) (
                 Token0:
             } else {
                 Token1;
                 Quantize(new,old,pro,lev,kle->frmh.quantizer(0]);
                 HuffEncLev(lev, buf);
                 PutData(addr,pro,dst);
             mode(oct)=M_QUIT;
             break;
        switch(mode(oct)) (
        case M_STOP:
             StopCounters(mode,oct.mask.blk,x,y.octs):
             break;
        case M_QUIT:
            break:
        default:
             DownCounters (mode, oct, mask, blk);
             break;
    ) while (mode(oct):=M_QUIT);
}
/* index to quant and vice versa */
#define i2q(i) (float)i*HISTO_DELTA/(float)HISTO
#define q2i(q) Fix2Long(X2Fix(q*(float)HISTO_HISTO_DELTA))
    Function Name: LookAhead
                     Examine base of tree to calculate new quantizer value
    Description:
    Arguments: src - source channel memory
                  dst - destination memory (and old for videos) octs, size - octaves of decomposition and image dimensions
                  norms - base HVS weighted normals
                  calculates new quant
    Returns:
```

Tengineering: KlicsCode: CompPict: KlicsEnc.c

```
LookAhead(short *src.short *dst.float norms(5)[5].KlicsE kle)
∵oid
             x. y. sub, index, size(2)=(kle->seqh.sequence_size(0),kle->seqh.sequence_sh(HISTO), quact(HISTO), target:
             new(4), old(4), addr(4), zero(4)=(0.0.0.0):
    int
             quant:
    float
    for(index=0:index<HISTO:index++) (</pre>
         thrash(index)=0;
         quact(index)=0:
    for(y=0:y<s1ze(1):y+=2<<octs)
     for(x=0;x<slze(0);x+=2<<octs)
     for(sub=1:sub<4:sub++) (
         float q_thresh:
                 nz. no. oz. blk;
         int
         Boolean ozflag, origin, motion;
         GetAddr(addr.x,y,sub.octs-1,size,1<<octs);</pre>
         GetData(addr.new.src);
         GetData(addr.oid.dst):
         nz=Decide(new):
         oz=Decide(old);
         no=DecideDelta(new.old);
         ozflag=kle->encd.intra || BlkZero(old);
         origin=nz<=no;
         motion=(nz+oz)>>octs <= no;
          q_thresh=(float)nz/DecideDouble(norms[1][1]);
          if (ozflag || origin) (
                      qt=GuessQuantize(new,zero,norms[1][0]);
              float
              q_thresh=q_thresh<qt?q_thresh:qt;
          ) else (
              float qt=GuessQuantize(new,old,norms[1][0]);
              q_thresh=q_thresh<qt?q_thresh:qt;
              if (!motion) (
                   qr=(float)no/DecideDouble(norms[1][2]);
                   q_thresh=q_thresh<qt?q_thresh:qt;
              )
          index=q2i(q_thresh);
          index=index<0?0:index>HISTG-1?HISTO-1:index:
          thresh(index)++;
      for(index=HISTO-1;index>=0;index--)
          quact(index)=thresh(index)*index+(index=HISTO-1?0:quact(index+1));
      /* buffer must be greater than bfp_in after this frame */
/* buffer must be less than buff_size+bfp_in */
      target=kle->encd.bpf_out*kle->encd.prevquact/kle->encd.prevbytes; /* previous
      index=1:
      while(index<HISTO && quact[index]/index>target) index++;
      quant=i2q(index):
      kle->encd.tmp_quant=(kle->encd.tmp_quant+quant)/2.0;
kle->encd.tmp_quant=i2q((index=q2i(kle->encd.tmp_quant))); /* forward and reve
      kle->encd.prevquact=quact(index)/(index==0?1:index);
  }
  /* Function Name: BaseNormals
```

Engineering: KlicsCode: CompPict: KlicsEnc.c

```
Calculates base HVS weighted normals
   Description:
   Arguments: norms - storage for normals
                weighted normals
    Returns:
        BaseNormals(float norms[5][3], KlicsE kle)
void
            base_norm(3)=(1.0,kle->encd.thresh,kle->encd.compare):
            norm, oct:
    int
    for (oct=0:oct<5:oct++)
        for (norm=0; norm<3; norm++)</pre>
                 norms(oct)(norm)=base_norm(norm)*kle->encd.base(oct)*(float)(1<<kl
}
    Function Name: Normals
                     Calculates HVS weighted normals @ quant
    Description: -
    Arguments: norms - storage for normals
                weighted normals and LPF bits
    Returns:
        Normals(float base_norms(5)[3], int thresh(5), int compare(5), KlicsE kle)
    int
             oct, i. norm;
    for(oct=0:oct<=kle->seqh.octaves(0):oct++) {
         norm=Fix2Long(X2Pix(base_norms(oct)(0)*kle->encd.tmp_quant));
         norm=norm<171:norm;
         for (i=0;0!=(norm6-3);i++)
                 norm=norm>>1;
         switch(norm) (
         case 1:
             kle->frmh.quantizer(oct)=i:
             break;
         case 2:
             kle->frmh.quantizer(oct)=i+1;
             break;
         case 3:
         case 4:
             kle->frmh.quantizer(oct)=i+2;
         thresh(oct) =Fix2Long(X2Fix(DecideDouble(base_norms[oct][1]*kle->encd.tmp_q
         compare[oct]=Fix2Long(X2Fix(DecideDouble(base_norms(oct)[2]*kle->encd.tmp_
    kle->frmh.quantizer(0)=kle->frmh.quantizer(0)<3?3:kle->frmh.quantizer(0);
/* minimum 4 bits of quant for lpf due to dynamic range problems */
Boolean KlicsFlags(KlicsE kle)
     Boolean skip=false;
     kle->encd.buffer-*kle->encd.bpf_in;
     kle->frmh.flags=0;
     if (kle->encd.buffer<0)
         kle->encd.buffer=0;
     if (kle->encd.intra)
         kle->frmh.flags |= KFH_INTRA:
     else
         if (skip=kle->encd.buf_sw && kle->encd.buffer>=kle->encd.buf_size)
             kle->frmh.flags |= KPH_SKIP;
     recurn(skip);
```

Engineering: KlicsCode: CompPict: KlicsEnc. c

```
Function Name: KlicsEncode
                    Encode a frame from YUV (de)transformed image
   Description:
   Arguments: src - source image(s)
                dst - transformed destination memory (and old for videos)
long
        KlicsEncode(short 'src(3), short 'dst(3), KlicsE kle)
   float
            base_norms(5)(3);
   ınt
            channel. chresh(5). compare(5);
   Buf
            buf=&kle->buf;
   buf_winit(buf)
   if (KlicsFlags(kle))
        kle->frmh.length=0:
    else (
        for(channel=0:channel<kle->seqh.channels:channel++) {
                     size(2)=(kle->seqh.sequence_size(0)>>(channel==0?0:kle->seqh.s
                             kle->seqh.sequence_size(1)>>(channel==0?0:kle->seqh.su
                         area=size(0)*size(1). octs=kle->seqh.octaves(channel==0?0:
            switch(kle->seqh.wavelet) (
            case WT_Haar:
                HaarForward(src(channel), size, octs);
                break:
            case WT_Daub4:
                Daub4Forward(src(channel), size.octs):
                break;
            ١
        BaseNormals(base_norms, kle);
        if (kle->encd.auto_q && !kle->encd.intra)
   LookAhead(src[0],dst[0],base_norms.kle);
        else
        kle->encd.tmp_quant=kle->encd.quant;
Normals(base_norms,thresh,compare,kle);
        for(channel=0;channel<kle->seqh.channels:channel++) (
                     size(2)=(kle->seqh.sequence_size(0)>>(channel==0?0:kle->seqh.s
            int
                        kle->such.sequence_size(1)>>(channel==0?0:kle->seqh.sub_sa
                     octs=kle->seqh.octaves(channel==0?0:1);
            if (kle->encd.intra)
                KLZERO(dst(channel), size(0)*size(1));
            if (channel==0) KlicsEncy(src[channel],dst[channel],octs.size.thresh.c
            else KlicsEncUV(src(channel),dst(channel),dcts.size,thresh.compare.kle
        buf_flush(buf);
        kle->frmh.length=buf_size(buf);
        kle->encd.buffer+=kle->frmh.length;
        if (!kle->encd.intra)
            kle->encd.prevbytes=kle->frmh.length:
   return(kle->frmh.length);
```

Implementation notes :

Possible developments:

KlicsFrameHeader.quantizer

```
Engineering: KlicsCode: CompPict: KlicsHeader.h
·····
    © Copyright 1993 KLICS Limited
   All rights reserved.
   Written by: Adrian Lewis
 Sequence and frame headers for Klics-Encoded files
   High byte first
typedef struct (
   unsigned short description_length; /* Fixed
                                                      - Size of this or parent struc
    unsigned char version_number[2]; /* Fixed
                                                      - Version and revision numbers
| KlicsHeader:
typedef struct (
   KlicsHeader head;
                                          /* Fixed
                                                      - Size and version of this str
   unsigned short sequence_size[3];
                                         / Source
                                                     - Cuminance dimensions and num
                                         /* Source · - Number of channels: 3 - YUV,
   unsigned char channels:
   unsigned char sub_sample(2); unsigned char wavelet;
                                                     - UV sub-sampling in X and Y d
- Wavelet used: 0 - Haar, 1 -
Bit precision for transform
                                         /* Source
/* Source
                                         /* Source
/* Source
   unsigned char unsigned char
                    precision:
                                                      - Number of octaves Y/UV (maxi)
                     octaves[2]:
    unsigned char reserved(3);
                                         /* Fixed
                                                     - Reserved for future use "/
} RlicsSeqHeader;
typedef struct (
    KlicsHeader head:
                                         /* Fixed
                                                      - Size and version of this str
                                         /* Calc
/* Calc
   unsigned long
                   length;
                                                      - Length of frame data (bytes)
                    frame_number;
    unsigned long
                                                      - Prame number intended for se-
                                                      - Bitfield flags: 0 - frame sk
- Quantiser shift values(octav
    unsigned char
                    flags;
                                         /* Calc
                                         /* Calc
                    quantizer[5];
   unsigned char
                                         /* Fixed
                                                      - Reserved for future use */
    unsigned short reserved;
) KlicsFrameHeader:
#define KFH_SKIP
                    0x1
≠define KFH_INTRA
                    0x2
```

QuickTime Hust have KlicsFrameHeader.length set to a valid number Sun Hust have KlicsSegHeader in data stream

Different values for UV and GH. HG.GG sub-bands are not currently suppo

Currently contains shift rather than step-size

```
Engineering:Kl:csCode:Kl:cs Codec:Kl:csEncode.r
    KlicsEncode resource file
*include 'Types.r'
*include 'MPWTypes.r'
*include 'ImageCodec.r'
   Klics Compressor included into the applications resource file here
*define klicsCodecFormatName
                                  "Klics"
#define klicsCodecFormatType
                                  'klic'
1.
    This structure defines the capabilities of the codec. There will
    probably be a tool for creating this resource, which measures the performance
    and capabilities of your codec.
resource 'cdci' (129, 'Klics CodecInfo', locked) (
    klicsCodecFormatName,
                                                       /* name of the codec TYPE ( da
    1.
                                                      /* version */
    1.
                                                      /* revision */
    'klic'.
                                                      /* who made this codec */
    Ο,
    codecInfoDoes32!codecInfoDoes8!codecInfoDoesTemporal.
                                                              /* depth and etc suppo
                                                      /* which data formats do we up.
/* compress accuracy (0-25) (
    codecInfoDepth24/codecInfoSequenceSensitive,
    100.
    100,
                                                      /* decompress accuracy (0-255)
    0,
                                                      /* millisecs to compress 320x2
    O,
                                                      /* millisecs to decompress 320.
    0,
                                                      /* compression level (0-255) (
    ٥,
    32.
                                                      /* minimum height */
                                                      /* minimum width */
    32.
    С.
    ٥.
1 :
resource 'thing' (128, 'Klics Compressor', locked) (
    compressorComponentType.
    klicsCodecformatType.
    'klic',
    codecInfoDoes32!codecInfoDoes8!codecInfoDoesTemporal,
    ٥.
    cdec .
    128,
    'STR '.
    128,
    STR .
    129.
    'ICON'.
    128
);
resource 'STR ' (128) (
    "Klics Compress"
```

- 805 -

```
Engineering: KlicsCode: Klics Codec: KlicsDecode.r

    KlicsDecode resource file

=include 'Types.r'
*include 'MPWTypes.r'
*include 'ImageCodec.r'
  * Klics Compressor included into the applications resource file here
*define klicsCodecFormatName
                                                                                           'Klics'
*define klicsCodecFormatType
           This structure defines the capabilities of the codec. There will
           probably be a tool for creating this resource. Which measures the performance
           and capabilities of your codec.
 resource 'cdci' (129, 'Klics CodecInfo'.locked) (
                                                                                                                                                   /* name of the codec TYPE ( da
           klicsCodecFormatName,
                                                                                                                                                   /* version */
                                                                                                                                                   /* revision */
                                                                                                                                                    /* who made this codec */
            'klic'.
           codecInfoDoes32|codecInfoDoes16|codecInfoDoes8|codecInfoDoesTemporal|codecInfo
                                                                                                                                                   /* which data formats do we un-
/* compress accuracy (0-255) (
/* decompress accuracy (0-255)
           codecInfoDepth24;codecInfoSequenceSensitive,
            100.
            100,
                                                                                                                                                   /* millisecs to compress 320x2
/* millisecs to decompress 320
           Ō,
            C.
                                                                                                                                                   /* compression level (0-255) (
           0.
            c.
                                                                                                                                                   /* minimum height */
           32.
                                                                                                                                                   /* minimum width */
            32.
           C.
1:
 resource 'thing' (130, 'Klics Decompressor'.locked) (
           decompressorComponentType.
            klicsCodecFormatType.
            'klic'
           \verb|codecInfoDoes32|| codecInfoDoes16|| codecInfoDoes8|| codecInfoDoes7emporal|| codecInfoDoes7emporal|| codecInfoDoes8|| codecInfoDoes7emporal|| codecInfoDoes8|| codecInfoDoes
            'cdec',
           128,
'STR ',
            130.
             'STR ',
            131,
             'ICON',
            130
 ):
 resource 'STR ' (130) {
```

CLAIMS

10

15

20

25

30

35

WE CLAIM:

1. A method of transforming a sequence of input digital data values into a first sequence of transformed 5 digital data values and of inverse transforming a second sequence of transformed digital data values into a sequence of output digital data values, said sequence of input digital data values comprising a boundary subsequence and a non-boundary subsequence, comprising the steps of:

running a number of said input digital data values of said boundary subsequence through a low pass boundary forward transform perfect reconstruction digital filter and through a high pass boundary forward transform perfect reconstruction digital filter to produce a first subsequence of said first sequence of transformed digital data values, said first subsequence of said first sequence of transformed digital data values comprising interleaved low and high frequency transformed digital data values:

running a number of said input digital data values of said non-boundary subsequence through a low pass non-boundary forward transform perfect reconstruction digital filter and also through a high pass non-boundary forward transform perfect reconstruction digital filter to produce a second subsequence of said first sequence of transformed digital data values, said second subsequence of said first sequence of transformed digital data values comprising interleaved low and high frequency transformed digital data values, said low pass boundary forward transform perfect reconstruction digital filter having a fewer number of coefficients than said low pass non-boundary forward transform perfect reconstruction digital filter, said high pass boundary forward transform perfect reconstruction digital filter having a fewer number of coefficients

WO 94/23385 PCT/GB94/00677

- 808 -

than said high pass non-boundary forward transform perfect reconstruction digital filter;

5

10

15

20

25

30

converting said first sequence of transformed digital data values into said second sequence of transformed digital data values, said second sequence of transformed digital data values comprising a first subsequence of said second sequence of transformed digital data values and a second subsequence of said second sequence of transformed digital data values;

running a number of said first subsequence of said second sequence of transformed digital data values through an interleaved boundary inverse transform perfect reconstruction digital filter to produce at least one output digital data value;

running a number of said second subsequence of said second sequence of transformed digital data values through a first interleaved non-boundary inverse transform perfect reconstruction digital filter to produce output digital data values; and

running a number of said second subsequence of transformed digital data values through a second interleaved non-boundary inverse transform perfect reconstruction digital filter to produce output digital data values, said output digital data values produced by said interleaved boundary inverse transform perfect reconstruction digital filter, said first interleaved non-boundary inverse transform perfect reconstruction digital filter, and said second interleaved non-boundary inverse transform perfect reconstruction digital filter comprising a subsequence of said output digital data values of said sequence of output digital data values.

 The method of Claim 1, wherein said low pass boundary forward transform perfect reconstruction digital
 filter has X coefficients and wherein said low pass nonboundary forward transform perfect reconstruction digital

filter has Y coefficients, Y being greater than X, said X coefficients of said low pass boundary forward transform perfect reconstruction digital filter being chosen so that said low pass boundary forward transform perfect 5 reconstruction digital filter outputs a transformed digital data value H₀ when the low pass boundary forward perfect transform reconstruction digital filter operates on input digital data values ID₀-ID_{x-1} adjacent said boundary, said transformed digital data value Ho being substantially equal 10 to what the output of the low pass non-boundary forward transform perfect reconstruction digital filter would be were the low pass non-boundary forward perfect reconstruction digital filter to operate on $ID_0-ID_{\chi-1}$ as well as Y-X additional input digital data values outside 15 said boundary, said additional input digital data values having preselected values.

- 3. The method of Claim 2, wherein Y-X=1, wherein there is one additional input digital data value ID_{-1} , and wherein ID_{-1} is preselected to be substantially equal to ID_{0} .
 - 4. The method of Claim 2, wherein Y-X=1, wherein there is one additional input digital data value ID_{-1} , and wherein ID_{-1} is preselected to be substantially equal to zero.
- 5. The method of Claim 1, wherein said sequence of input digital data values is a sequence of digital data values associated with pixels of either a row or a column of a two dimensional image, said boundary of said sequence of input digital data values corresponding with either a start or an end of said row or said column.
 - 6. The method of Claim 1, wherein said sequence of input digital data values is a sequence of digital data values associated with an audio signal.

- 7. The method of Claim 1, wherein said low and high pass non-boundary forward transform perfect reconstruction digital filters are forward transform quasi-perfect reconstruction filters which have coefficients which 5 approximate the coefficients of true forward transform perfect reconstruction filters.
- 8. The method of Claim 1, wherein said low and high pass non-boundary forward transform perfect reconstruction digital filters are both four coefficient quasi-Daubechies filters the coefficients of which approximate the coefficients of true four coefficient Daubechies filters.
 - 9. The method of Claim 8, wherein one of said four coefficient quasi-Daubechies filters has the coefficients 11/32, 19/32, 5/32 and 3/32 independent of sign.
- 10. The method of Claim 1, wherein said low pass nonboundary forward transform perfect reconstruction digital filter is a four coefficient quasi-Daubechies filter H of the form:

$$H_n = aID_{2n-1} + bID_{2n} + cID_{2n+1} - dID_{2n+2}$$

20 n being a positive integer, ${\rm ID_0}\text{-}{\rm ID_m}$ being input digital data values, m being a positive integer, ${\rm ID_0}$ being the first input digital data value in said sequence of input digital data values, and wherein said low pass boundary forward transform perfect reconstruction digital filter is a three 25 coefficient digital filter of the form:

$$H_0 = aID_{-1} + bID_0 + cID_1 - dID_2$$

 ${\rm ID}_{-1}$ being a predetermined input digital data value outside said boundary and having a preselected value.

11. The method of Claim 10, wherein said high pass

WO 94/23385 PCT/GB94/00677

- 811 -

non-boundary forward transform perfect reconstruction digital filter is a four coefficient quasi-Daubechies filter of the form:

$$G_n = dID_{2n-1} + cID_{2n} - bID_{2n+1} + aID_{2n+2}$$

5 n being a positive integer, and wherein said high pass boundary forward transform perfect reconstruction digital filter is a three coefficient digital filter of the form:

$$G_0 = dID_{-1} + cID_0 - bID_1 + aID_2$$

dID_1 having a preselected value.

- 10 The method of Claim 11, wherein: a + b + c - d is 12. substantially equal to 1, wherein a - b + c + d is substantially equal to 0, and wherein ac - bd is substantially equal to zero.
- The method of Claim 12, wherein: a=11/32, 13. 15 b=19/32, c=5/32 and d=3/32.
 - The method of Claim 11, wherein said interleaved boundary inverse transform perfect reconstruction digital filter is a two coefficient digital filter of the form:

$$OD_0 = 4(b-a)H_0 + 4(c-d)G_0$$

- 20 wherein ODo is an output digital data value of said sequence of output digital data values, wherein G_0 is the output of said high pass boundary forward transform perfect reconstruction digital filter when the high pass boundary forward transform perfect reconstruction digital
- 25 filter operates on input digital data values IDo, ID1 and ID, adjacent said boundary, and wherein Ho is the output of said low pass boundary forward transform perfect reconstruction digital filter when the low pass boundary

forward transform perfect reconstruction digital filter operates of input digital data values ID, ID, and ID, adjacent said boundary.

The method of Claim 14, wherein one of said first 5 and second interleaved non-boundary inverse transform perfect reconstruction digital filters is of the form:

$$D_{2n+1} = 2(cH_n - bG_n + aH_{n+1} + dG_{n+1})$$

n being a non-negative integer, and wherein the other of said first and second interleaved non-boundary inverse 10 perfect reconstruction digital filters is of the form:

$$D_{2n+2} = 2(-dH_n + aG_n + bH_{n+1} + cG_{n+1})$$

n being a non-negative integer, wherein $\mathbf{H}_n,~\mathbf{G}_n,~\mathbf{H}_{n+1}$ and \mathbf{G}_{n+1} comprise a subsequence of said second sequence of transformed digital data values.

- 15 The method of Claim 1, wherein said low pass nonboundary forward transform perfect reconstruction digital filter is a four coefficient quasi-Daubechies filter having the coefficients: 11/32, 19/32, 5/32 and -3/32, and wherein said high pass non-boundary forward transform perfect 20 reconstruction digital filter is a four coefficient quasi-Daubechies filter having the coefficients: 3/32, 5/32, -19/32 and 11/32.
- The method of Claim 1, wherein said low and high pass non-boundary forward transform perfect reconstruction 25 digital filters are chosen from the group consisting of: true six coefficient Daubechies filters and quasi-Daubechies filters, the coefficients of the quasi-Daubechies filters approximating the coefficients of true six coefficient Daubechies filters.

- 813 -

18. The method of Claim 1, further comprising the steps of:

5

10

encoding said first sequence of transformed digital data values into an encoded sequence; and decoding said encoded sequence of digital data values into said second sequence of transformed digital data values and supplying said second sequence of transformed digital data values to said interleaved boundary inverse transform perfect reconstruction digital filter, said first interleaved non-boundary inverse transform perfect reconstruction digital filter, and said second interleaved non-boundary inverse transform perfect reconstruction digital filter.

15 19. The method of Claim 18, further comprising the step of:

quantizing each of said digital data values in said first sequence of transformed values before said encoding step.

- 20. The method of Claim 1, wherein each of said input digital data values of said sequence of input digital data values is stored in a separate memory location, and wherein some of said memory locations are overwritten in a sequence with said sequence of transformed digital data values as 25 said digital data input values are transformed into said transformed digital data values.
- 21. A method of transforming a sequence of input digital data values into a sequence of transformed digital data values, said sequence of input digital data values

 30 comprising a boundary subsequence and a non-boundary subsequence, comprising the steps of:

running a number of said input digital data values of said boundary subsequence through a low pass boundary forward transform perfect reconstruction

WO 94/23385 PCT/GB94/00677

- 814 -

digital filter and through a high pass boundary forward transform perfect reconstruction digital filter to produce a first subsequence of said sequence of transformed digital data values, said first subsequence of said sequence of transformed digital data values comprising interleaved low and high frequency transformed digital data values; and

5

10

15

20

25

30

35

running a number of said input digital data values of said non-boundary subsequence through a low pass non-boundary forward transform perfect reconstruction digital filter and also through a high pass non-boundary forward transform perfect reconstruction digital filter to produce a second subsequence of said sequence of transformed digital data values, said second subsequence of said sequence of transformed digital data values comprising interleaved low and high frequency transformed digital data values, said low pass boundary forward transform perfect reconstruction digital filter having a fewer number of coefficients than said low pass non-boundary forward transform perfect reconstruction digital filter, said high pass boundary forward transform perfect reconstruction digital filter having a fewer number of coefficients than said high pass nonboundary forward transform perfect reconstruction digital filter.

22. A method, comprising the steps of:

generating a sub-band decomposition having a plurality of octaves, a first of said plurality of octaves comprising at least one first digital data value, a second of said plurality of octaves comprising at least one second digital data value;

calculating a sum of the absolute values of said at least one first digital data value;

determining if said at least one first digital data value is interesting using a first threshold

5

15

limit;

calculating a sum of the absolute values of said at least one second digital data value; and determining if said at least one second digital data value is interesting using a second threshold limit.

23. A method of traversing a tree decomposition, said tree decomposition comprising a plurality of transformed data values, each of said plurality of transformed data values having a unique address identified by coordinates X and Y, comprising the step of:

calculating at least four transformed data value addresses by incrementing a count, the count comprising one bit Cl_{χ} in the X coordinate and one bit Cl_{γ} in the Y coordinate, to generate said at least four transformed data value addresses.

24. A method, comprising the step of:

determining an address of a transformed data value in a tree decomposition by shifting a value a number of times, 20 said tree decomposition having a number of octaves, said transformed data value being in one of said octaves, said number of times being at least dependent upon said one octave.

- 25. A method, comprising the step of:
- determining an address of a transformed data value in a tree decomposition by multiplying a value by a factor, said tree decomposition having a number of octaves, said transformed data value being in one of said octaves, said factor being at least dependent upon said one octave.
- 26. A method, comprising the step of:

 determining an address of a transformed data value in
 a tree decomposition by shifting a value a number of times,
 said tree decomposition having a number of frequency sub-

bands, said transformed data value being in one of said frequency sub-bands, said number of times being at least dependent upon said frequency sub-band.

27. A method, comprising the step of:

- determining an address of a transformed data value in a tree decomposition by performing a logical operation upon a value, said tree decomposition having a number of frequency sub-bands, said transformed data value being in one of said frequency sub-bands, said logical operation performed being at least dependent upon said one frequency sub-band.
 - 28. The method of Claim 27, wherein said logical operation is a bit-wise logical AND operation.
- 29. A method for determining a low pass quasi-perfect
 15 reconstruction filter and a high pass quasi-perfect
 reconstruction filter from a wavelet function, said low
 pass quasi-perfect reconstruction filter having a plurality
 of coefficients, said high pass quasi-perfect
 reconstruction filter having a plurality of coefficients,
 20 comprising the steps of:

determining a low pass wavelet digital filter and a high pass wavelet digital filter from said wavelet function, said low pass wavelet digital filter having a plurality of coefficients, said high pass wavelet digital 25 filter having a plurality of coefficients;

choosing the coefficients of said low pass quasiperfect reconstruction digital filter to be fractions such
that when a sequence of data values having values of 1 is
processed by said low pass quasi-perfect reconstruction
30 digital filter the output of said low pass quasi-perfect
reconstruction digital filter is exactly a power of 2; and

choosing the coefficients of the high pass quasiperfect reconstruction digital filter to be fractions such that when a sequence of data values having values of 1 is WO 94/23385 PCT/GB94/00677

- 817 -

processed by said high pass quasi-perfect reconstruction digital fifter the output of said high pass quasi-perfect reconstruction digital filter is exactly 0, whereby each of the plurality of coefficients of said low pass quasi
5 perfect reconstruction digital filter is substantially identical to a corresponding one of said plurality of coefficients of said low pass wavelet digital filter, and whereby each of the plurality of coefficients of said high pass quasi-perfect reconstruction digital filter is

10 substantially identical to a corresponding one of said plurality of coefficients of said high pass wavelet digital filter.

30. A method of estimating a compression ratio of a number of original data values to a number of compressed 15 data values at a value of a quality factor Q, comprising the steps of:

examining a first block of transformed data values of a tree, said first block being one of a number of lowest frequency blocks of a high pass component sub-band, said 20 tree being part of a sub-band decomposition; and

determining a value of said quality factor Q at which said data values of said first block would be converted into compressed data values, and not determining a value of said quality factor Q at which any other block of data 25 values of said tree would be converted into a number of compressed data values.

- 31. The method of Claim 30, wherein said number of original data values represents a frame of an image.
- 32. The method of Claim 31, further comprising the 30 step of:

determining a number of lowest frequency blocks of said high pass component sub-band which would be converted into compressed data values given a value of said quality factor Q.

PCT/GB94/00677 WO 94/23385

33. A method of transforming a sequence of image data

- 818 -

filtering said sequence of image data values using a quasi-perfect reconstruction filter to generate a 5 decomposition having a plurality of octaves, said quasiperfect reconstruction filter having six coefficients.

values, comprising the step of:

- 34. The method of Claim 33, wherein said six coefficients are selected from the group consisting of: 30/128, 73/128, 41/128, 12/128, 7/128 and 3/128, 10 irrespective of sign.
 - A method of detecting motion in a tree decomposition, said tree decomposition comprising a plurality of octaves of blocks of data values, comprising the steps of:
- comparing data values of a first block in an octave 15 with data values of a second block in said octave; and generating a token indicating motion based on said comparing.
 - 36. A method, comprising the steps of:
- generating a sub-band decomposition having a plurality 20 of octaves, a first of said plurality of octaves comprising at least one first digital data value, a second of said plurality of octaves comprising at least one second digital data value:
- determining if said at least one first digital data 25 value is interesting using a first threshold limit; and determining if said at least one second digital data value is interesting using a second threshold limit.
 - 37. A method, comprising the steps of:
- generating a sub-band decomposition of a first frame 30 having a plurality of octaves, a first of said plurality of octaves comprising at least one first digital data value, a

second of said plurality of octaves comprising at least one second digital data value;

generating a sub-band decomposition of a second frame having a plurality of octaves, a first of said plurality of octaves comprising at least one first digital data value, a second of said plurality of octaves comprising at least one second digital data value;

comparing said first digital data value of said first frame with said first digital data value of said second 10 frame using a first threshold compare; and

comparing said second digital data value of said first frame with said second digital data value of said second frame using a second threshold compare.

38. A method, comprising the steps of:

reading a sequence of data values from a plurality of memory locations, each of said data values being stored in a separate one of said plurality of memory locations; and

overwriting some of said memory locations in a sequence as said data values are transformed into a 20 sequence of transformed data values of a sub-band decomposition.

39. A method, comprising the steps of:

performing a function on a plurality of data values of a new block to generate a first output value, said new 25 block being a block of data values of a sub-band decomposition of a new frame;

performing said function on a plurality of numbers to generate a second output value, each of said numbers substantially equalling a difference of a data value in 30 said plurality of data values of said new block and a corresponding data value in a corresponding plurality of data values of an old block, said old block being a block of data values of a sub-band decomposition of an old frame; and

generating a token if said first output value has a

predetermined relationship with respect to said second output value.

- 40. The method of Claim 39, wherein said token is a SEND_STILL token.
- 5 41. A method, comprising the steps of:

performing a function on a plurality of data values of a new block to generate a corresponding plurality of output values, said new block being a block of data values of a sub-band decomposition;

comparing each of said plurality of output values with a predetermined number; and

generating a token if substantially all of said output values have a predetermined relationship with respect to said predetermined number.

- 15 42. The method of Claim 41, wherein said token is a VOID token.
 - 43. A method, comprising the steps of:

subtracting each one of a plurality of data values of a new block with a corresponding one of a plurality of data 20 values of a old block to generate a corresponding plurality of output values, said new block being a block of data values of a sub-band decomposition of a new frame, said old block being a block of data values of a sub-band decomposition of a old frame;

comparing each of said plurality of output values with a predetermined number; and

generating a token if substantially all of said output values have a predetermined relationship with respect to said predetermined number.

30 44. The method of Claim 43, wherein said token is a VOID token.

- 45. A method, comprising the steps of:

 determining an absolute value for each of a plurality
 of data values of a block of a sub-band decomposition;
 determining a sum of said absolute values; and
 generating a token based on a comparison of said sum
 with a predetermined number.
 - 46. The method of Claim 45, wherein said token is a VOID token.
 - 47. A method, comprising the steps of:
- processing a sequence of first image data values using a low pass forward transform perfect reconstruction digital filter and a high pass forward transform perfect reconstruction digital filter to create a first sequence of transformed data values, said low pass forward transform perfect reconstruction digital filter and said high pass forward transform perfect reconstruction digital filter each having coefficients chosen from a first group of coefficients independent of sign;

converting said first sequence of transformed data 20 values into a second sequence of transformed data values; and

using digital circuitry to process said second sequence of transformed data values using a low pass inverse transform perfect reconstruction digital filter and a high pass inverse transform perfect reconstruction digital filter into a sequence of second image data values, said low pass inverse transform perfect reconstruction digital filter and said high pass inverse transform perfect reconstruction digital filter each having coefficients chosen from a second group of coefficients independent of sign.

48. The method of claim 47, wherein said digital circuitry used to process said second sequence of transformed data values is a digital computer having a

WO 94/23385 PCT/GB94/00677

- 822 -

microprocessor.

- 49. The method of claim 47, wherein at least one of the coefficients in said first group of coefficients is not contained in said second group of coefficients.
- 5 50. The method of claim 47, wherein said first group of coefficients has a different number of coefficients than said second group of coefficients.
- 51. The method of claim 50, wherein said sequence of first image data values is a sequence of chrominance data 10 values.
- 52. The method of claim 50, wherein said low pass forward transform perfect reconstruction digital filter and said high pass forward transform perfect reconstruction digital filter each have four coefficients, and wherein said low pass inverse transform perfect reconstruction digital filter and said high pass inverse transform perfect reconstruction digital filter each have two coefficients.
- 53. The method of claim 52, wherein said sequence of first image data values is a sequence of chrominance data 20 values.
- 54. The method of claim 47, wherein each of said coefficients of said low pass inverse transform perfect reconstruction digital filter and said high pass inverse transform perfect reconstruction digital filter is selected from the group consisting of: 5/8, 3/8 and 1/8, independent of sign.
 - 55. The method of claim 47, wherein said converting step comprises the steps of:

encoding said first sequence of transformed data 30 values into a compressed data stream; and

decoding said compressed data stream into said second sequence of transformed data values.

- 56. A method comprising the step of using digital circuitry to process a sequence of image data values using 5 a low pass forward transform perfect reconstruction digital filter and a high pass forward transform perfect reconstruction digital filter to generate a sub-band decomposition, said low pass forward transform perfect reconstruction digital filter and said high pass forward transform perfect reconstruction digital filter each having four coefficients, each of said four coefficients being selected from the group consisting of: 5/8, 3/8 and 1/8, independent of sign.
- 57. The method of claim 56, wherein said digital
 15 circuitry comprises means for low pass forward transform
 perfect reconstruction digital filtering and for high pass
 forward transform perfect reconstruction digital filtering.
- 58. A method comprising the step of using digital circuitry to process a sequence of transformed data values 20 of a sub-band decomposition using an odd inverse transform perfect reconstruction digital filter and an even inverse transform perfect reconstruction digital filter, said odd inverse transform perfect reconstruction digital filter and said even inverse transform perfect reconstruction digital filter and 5 filter each having four coefficients, each of said four coefficients being selected from the group consisting of: 5/8, 3/8 and 1/8, independent of sign.
 - 59. The method of claim 58, wherein said digital circuitry is a digital computer having a microprocessor.
- 30 60. A method comprising the step of generating a compressed data stream indicative of a video sequence from a sub-band decomposition, said compressed data stream

comprising a first data value, a first token, a second data value, and a second token, said first token being indicative of a first encoding method used to encode said first data value, said second token being indicative of a second encoding method used to encode said second data value, said first token consisting of a first number of bits and said second token consisting of a second number of bits.

- 61. The method of claim 60, wherein said first
 10 encoding method is taken from the group consisting of: SEND
 mode, STILL SEND mode, VOID mode, and STOP mode.
 - 62. The method of claim 60, wherein said first token is a single bit token.
 - 63. A method, comprising the steps of:
- forward transforming image data values to generate a first sequence of transformed data values of a first subband decomposition, said first sub-band decomposing having a first number of octaves;

converting said first sequence of transformed data 20 values into a second sequence of transformed data values;

using digital circuitry to inverse transforming said second sequence of transformed data values into a third sequence of transformed data values, said third sequence of transformed data values comprising a second sub-band

- 25 decomposition having a second number of octaves, said second number of octaves being smaller than said first number of octaves, said second sub-band decomposition having a low pass component, said low pass component of said second sub-band decomposition comprising data values
- 30 indicative of rows of data values of an image, said rows of said image extending in a first dimension, said image also having columns of said data values extending in a second dimension:

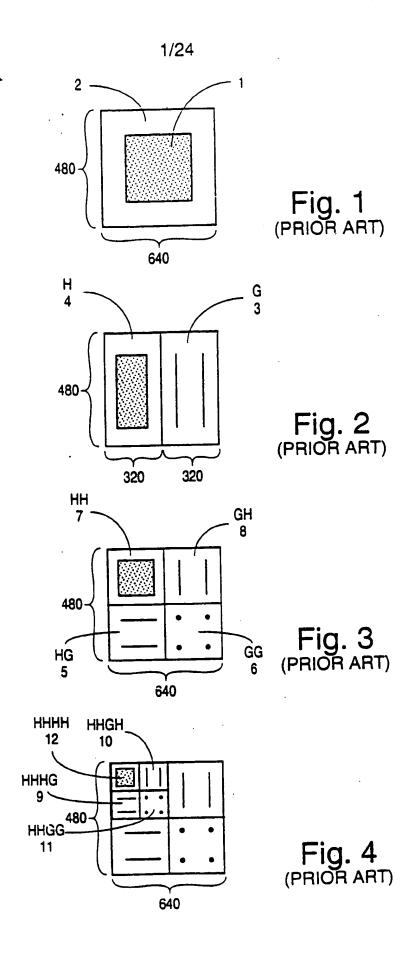
expanding said low pass component in said first

dimension using interpolation to generate an interpolated low pass component; and

expanding said interpolated low pass component in said second dimension by replicating rows of said data values of 5 said interpolated low pass component.

- 64. The method of claim 63, wherein said digital circuitry is a digital computer having a microprocessor.
- 65. The method of claim 63, wherein said converting step comprises the steps of:
- 10 encoding said first sequence of transformed data values into a compressed data stream comprising tokens and encoded data values; and

decoding said compressed data stream into said second sequence of transformed data values.



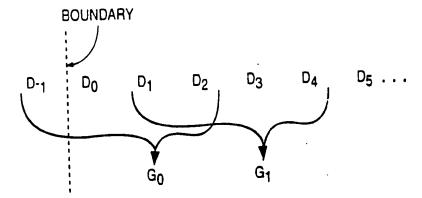


Fig. 5 (PRIOR ART)

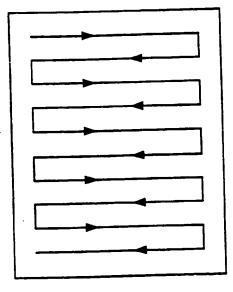


Fig. 6 (PRIOR ART)

3/24

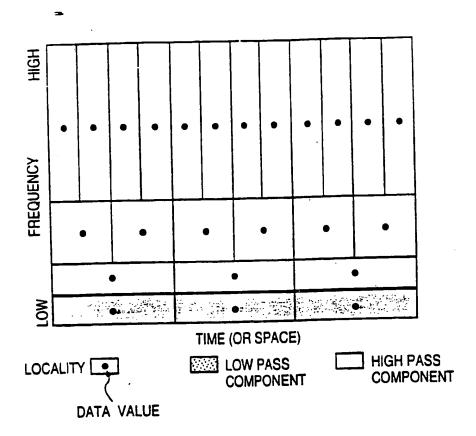
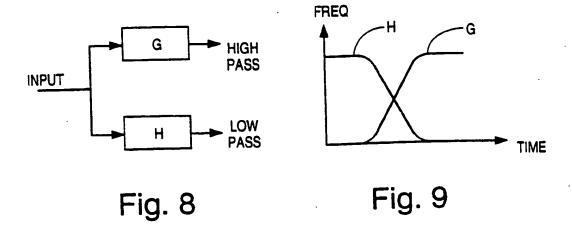
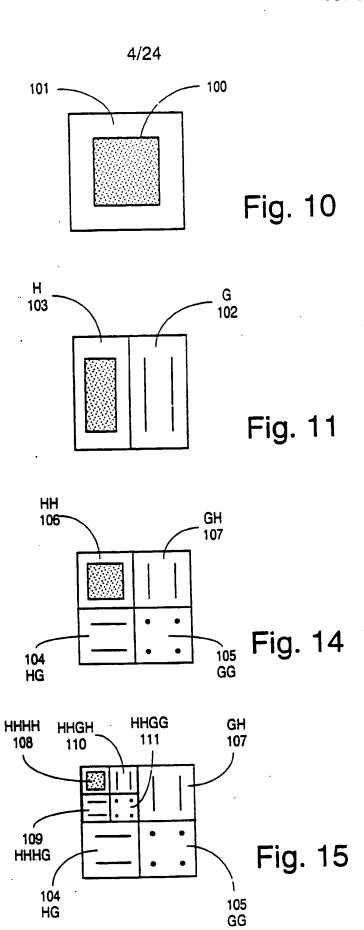


Fig. 7





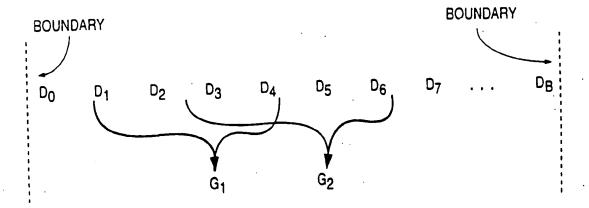


Fig. 12

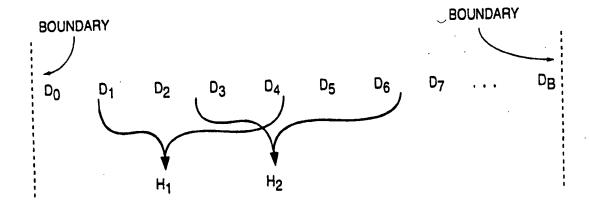


Fig. 13

	_												
	8	D ₀ B	0 ₁ B	D ₂ B	D _{3B}	D ₄ B	D ₅ B	D _{6B}	D7B	D8B	О9В	DAB	DeB
	V	D ₀ A	D1A	D2A	ОзА	D4A	D ₅ A	D ₆ A	D7A	D ₈ A	D ₉ A	DAA	Ова
	6	D ₀₉	D19	D ₂₉		D49			D79	D89	D ₉₉	DA9	DB9
	8	D ₀₈	D18	D ₂₈		D48			D78	D88	D38	DAB	DB8
	7	D ₀ 7	11 ₀		037				<i>L</i> ₀			D _A 7	
z	9	90 _Q	D ₁₆	D25 D26	D36	D46	D ₅₆	99 _O	9 ² 0	D86	D36	DA6	0BG
COLUM	2	D ₀₅	D ₁₅	D ₂₅	035	D45	D ₅₅	D ₆₅	D ₇₅	D ₈₅	095	DAS	DBS
	4	Dog	D ₁₄	024					D74	D ₈₄	D ₉₄	D _{A4}	DB4
	က	Dos	D ₁₃	D ₂₃	033	D43				Dgg	D ₉₃	DA3	DB3
	7	Dog	D ₁₂	022	D32	D42	D ₅₂	D ₆₃	D72	D82	D ₉₂	DA2	D _{B2}
	-	-0a	D11	021	D31		051	D ₆₁		D ₈₁	D ₉₁	DA1	DB1
	0	D00	D10	020	D30	D40	D ₅₀	09 ₀	070		D ₉₀	DAO	DB0
		0	-	2	က	4	В 0	≽	7	œ	6	⋖	6

Fig. 16

					J	COLUMN	-					
	0	-	2	3	4	2	9	_	8	6	<	<u>ه</u>
0	HH ₀₀	GH ₀₀	нН01	GH ₀₁	HH ₀₂	GH02	HH ₀₃	GH ₀₃	Ħ 8	GH ₀₄	HH05	GH ₀₅
-	HG ₀₀	GG00	HG01	6601	HG ₀₂	GG02	HG ₀₃	GG ₀₃ HG ₀₄	HG ₀₄	G G ₀₄	HG05	9099
2	HH ₁₀	GH ₁₀	∓	GH11	HH ₁₂	GH ₁₂	HH ₁₃	GH ₁₃	H 14	GH14	HH ₁₅	GH ₁₅
က	HG ₁₀	GG ₁₀	HG11	6611	HG ₁₂	6612	HG ₁₃	GG ₁₃	HG ₁₄	6614	HG ₁₅	6615
4	HH20	GH ₂₀	HH21	GH21	HH22	GH22	HH23	GH23	HH24	GH ₂₄	HH25	GH ₂₅
R O	HG20	GG20	HG21	6621	HG22	6622	HG23	6623	HG24	GG24	HG ₂₅	GG25
W 6	HH30	GH ₃₀	HH31	GH31	HH32	GH32	HH33	GH33	HH34	GH34	HH35	GH ₃₅
7	HG30	GG30	HG31	6631	HG32	GG32	HG33	6633	HG34	GG34	HG35	6635
æ	HH ₄₀	GH ₄₀	H.41	GH41	HH42	GH42	HH43	GH ₄₃	HH44	GH44	HH ₄₅	GH ₄₅
6	HG40	GG40	HG41	6641	HG42	GG42	HG43	GG43	HG44	GG 44	HG45	6645
∢	HH ₅₀	GH ₅₀	HH ₅₁	GH ₅₁	HH52	GH52	HH53	GH ₅₃	HH ₅₄	GH ₅₄	HH55	GH ₅₅
60	HG ₅₀	GG ₅₀	HG51	6651	HG52	6652	HG53	GG53	HG54	GG 54	HG55	6655

-ig. 17

	8	GH ₀₅	GG ₀₅	GH ₁₅	6615	GH25	GG25	GH ₃₅	GG 35	GH ₄₅	6645	GH ₅₅	6655	
	4	HHGH ₀₂	HG ₀₅	HHGG ₀₂	HG ₁₅	HHGH ₁₂	HG25	HHGG ₁₂	HG35	HHGH ₂₂	HG45	HHGG ₂₂	HG55	
	6	GH ₀₄	6604	GH ₁₄	6614	GH24	GG24	GH34	6634	GH ₄₄	GG 44	GH ₅₄	GG54	
	· 6 0	нинн ₀₂	HG ₀₄	ннн602	HG ₁₄	HHHH ₁₂	HG24	HHHG ₁₂ GH ₃₄	HG34	HHHH ₂₂	HG44	HHHG ₂₂	HG54	
	7	GH ₀₃	6603	GH ₁₃	Ġ G 13	GH23	6623	GH33	GG33	GH ₄₃	6643	GH ₅₃	6653	
	9	HHGH ₀₁ GH ₀₃	HG ₀₃	HHGG ₀₁ GH ₁₃	HG ₁₃	HHGH ₁₁ GH ₂₃	HG23	HHGG11 GH33	HG33	HHGH ₂₁	HG43	HHGG21 GH53	HG53	
	5	GH ₀₂	CG 02	GH ₁₂	6612	GH22	6622	GH32	6632	GH ₄₂	GG 42	GH ₅₂	6652	
COLUMN	4	нннн01	HG ₀₂	HHHG01 GH12	HG12	HHHH ₁₁ GH ₂₂	HG22	HHHG11 GH32	HG32	HHHH ₂₁	HG42	HHHG21 GH52	HG52	
	က	GH ₀ 1	6601	GH ₁₁	6611	GH21	6621	GH31	6631	GH41	6641	GH51	6651	
	2	HHGH ₀₀	HG ₀₁	HHGG00	HG ₁₁	HHGH ₁₀ GH ₂₁	HG21	HHGG10	HG31	HHGH ₂₀	HG41	HHGG20	HG51	
		GH ₀₀	00 00 00 00 00 00 00 00 00 00 00 00 00	GH ₁₀	6610	GH20	6620	GH30	G G30	GH ₄₀	GG40	GH ₅₀	6650	
	0	нинию ви	HG00 GG00	HHHG ₀₀ GH ₁₀	HG ₁₀ GG ₁₀	HHHH ₁₀ GH ₂₀	HG20	HHHG ₁₀ GH ₃₀	HG30 GG30	HHHH ₂₀ GH ₄₀	HG40	HHHG ₂₀ G	HG ₅₀	
		6	-	7	က	GE (2 ≤	9	7	80	တ	K	æ	

Fig. 18

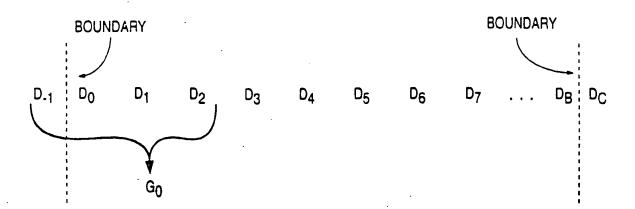


Fig. 19

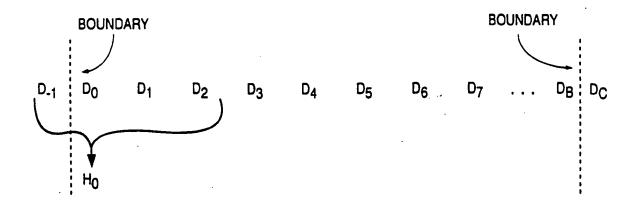
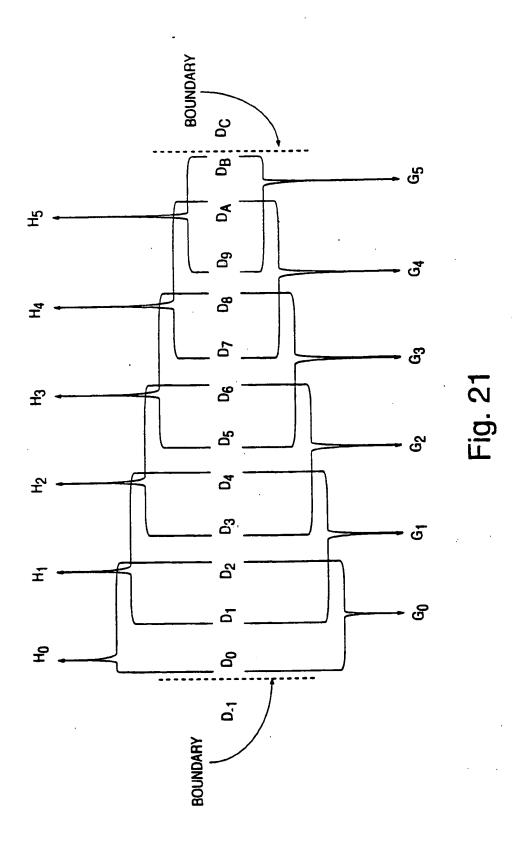


Fig. 20





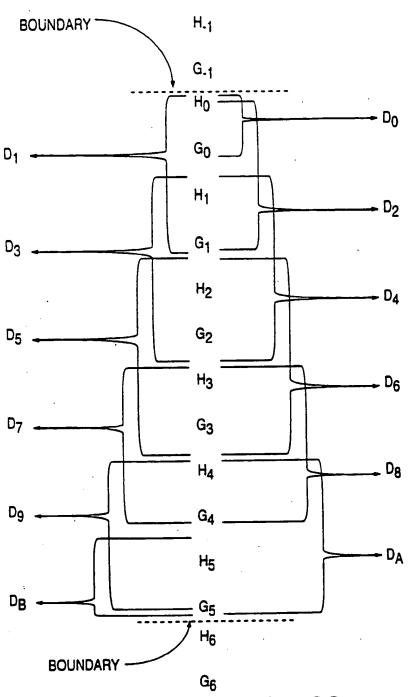


Fig. 22

12/24

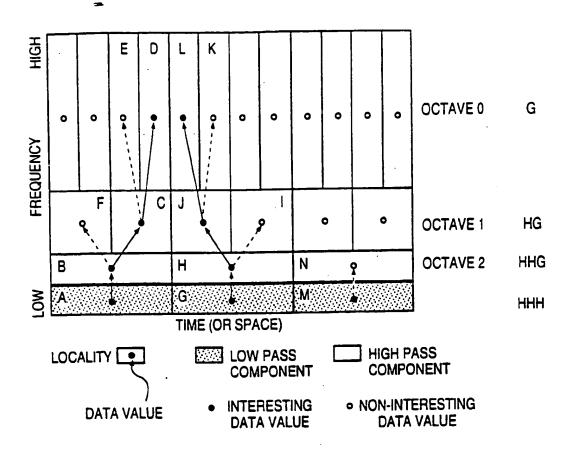
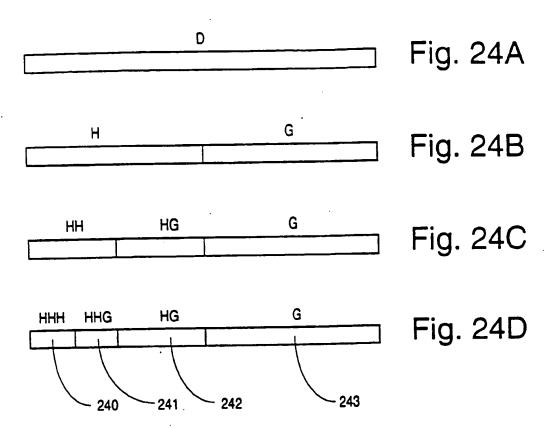
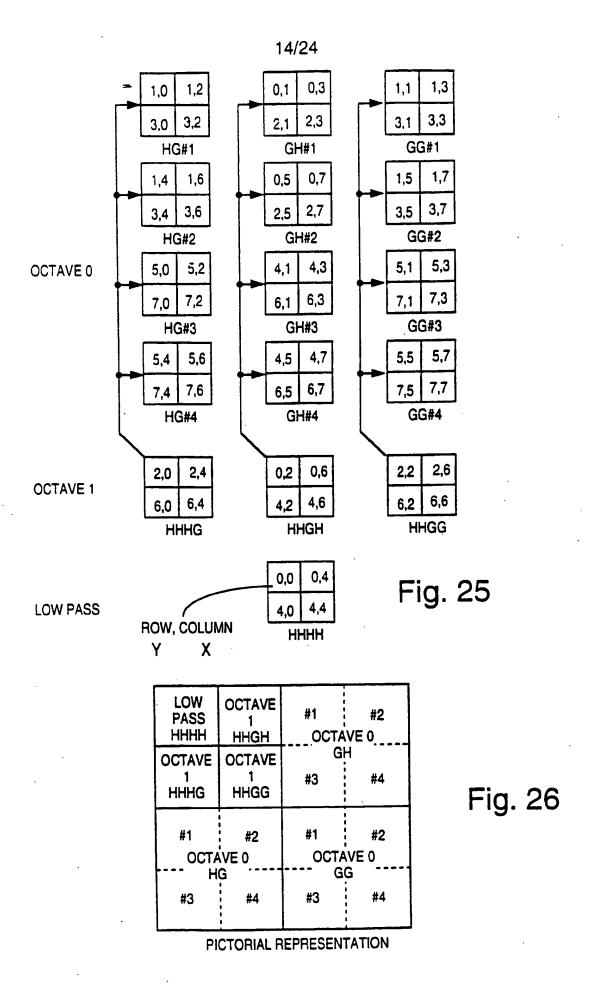


Fig. 23

13/24





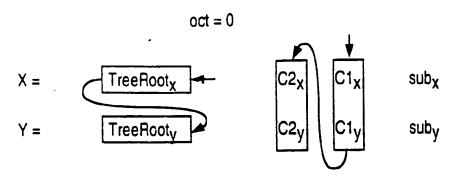


Fig. 27

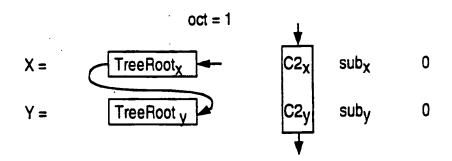
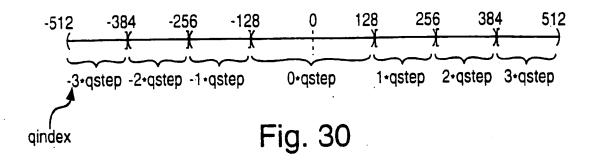


Fig. 28

_	sub-band	sub _X	suby
low pass	{ HH	0	0
	√ HG ← GH	0	1
high pass	{ GH	1	0
	(GG	1	1

Fig. 29

i



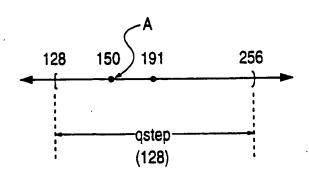
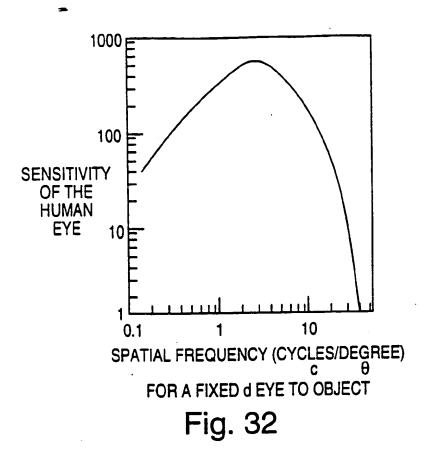


Fig. 31



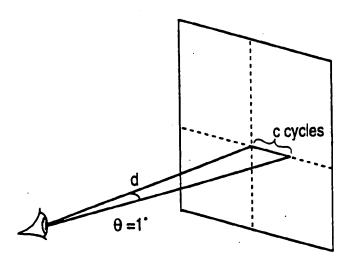


Fig. 33

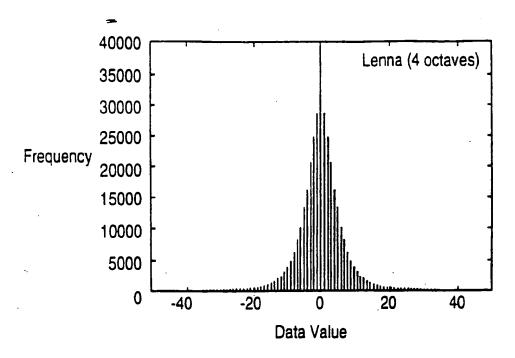
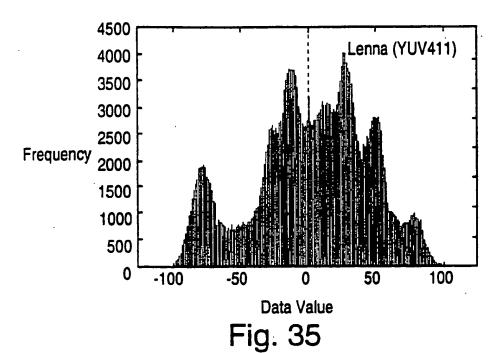
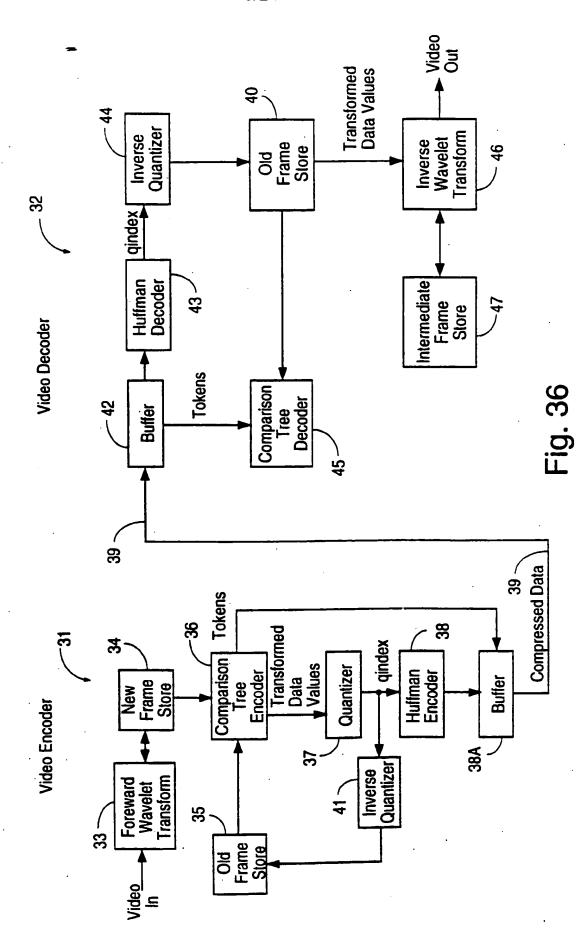


Fig. 34





MODES OF VIDEO ENCODER AND VIDEO DECODER

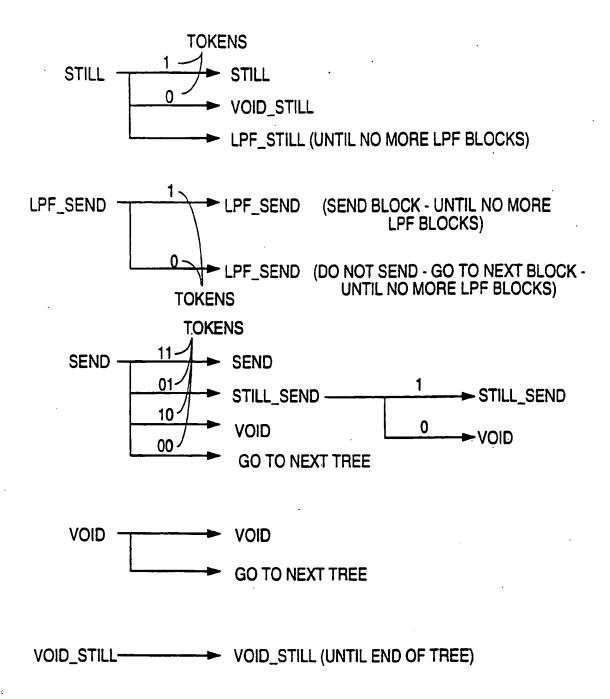
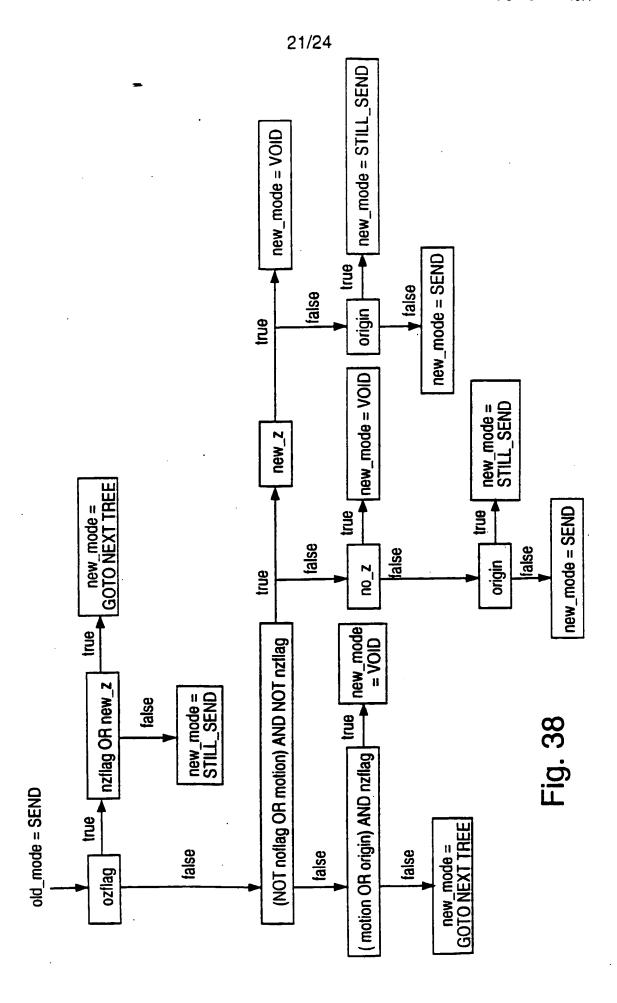


Fig. 37



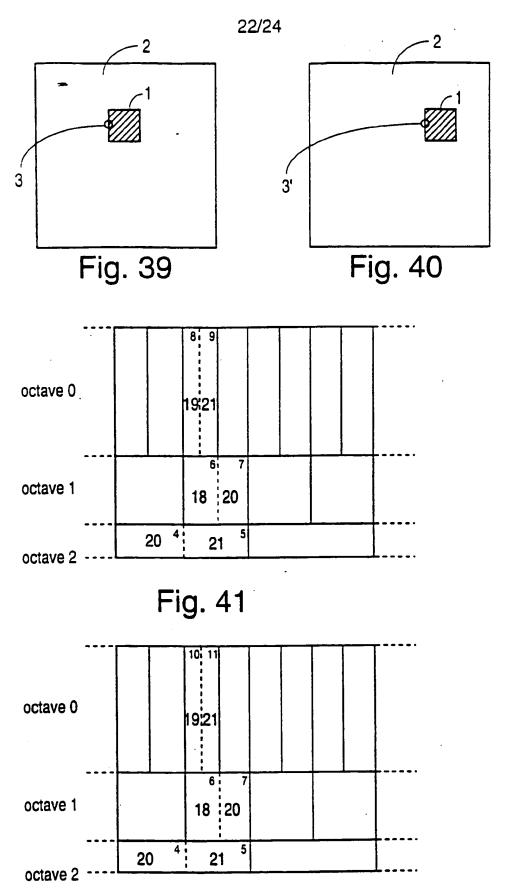


Fig. 42

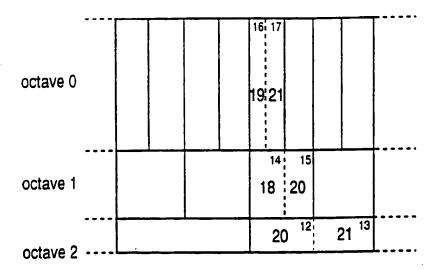
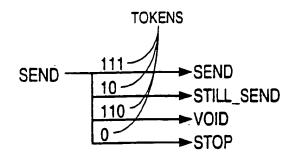


Fig. 43

VARIABLE - LENGTH TOKENS



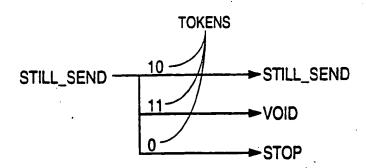


FIG. 44

This Page is Inserted by IFW Indexing and Scanning Operations and is not part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:
☐ BLACK BORDERS
☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
☐ FADED TEXT OR DRAWING
☐ BLURRED OR ILLEGIBLE TEXT OR DRAWING
☐ SKEWED/SLANTED IMAGES
☐ COLOR OR BLACK AND WHITE PHOTOGRAPHS
☐ GRAY SCALE DOCUMENTS
☐ LINES OR MARKS ON ORIGINAL DOCUMENT
☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
OTHER:

IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.